1-1-1998

# A Comparison of highly efficient iterative linear solvers

Akbar Chowdhury

# A Comparison of Highly Efficient Iterative Linear Solvers

by

**Akbar H. Chowdhury**

A Thesis submitted in partial fulfillment of the requirements for the Degree of Master of Science in Mechanical Engineering.

Approved by:

Professor_____
Dr. Amitabha Ghosh (Advisor)

Professor_____
Dr. Josef Török

Professor_____
Dr. Satish Kandlikar

Professor_____
Dr. Charles W. Haines

Department of Mechanical Engineering
College of Engineering
Rochester Institute of Technology
January, 1998

# A Comparison of Highly Efficient Iterative Linear Solvers

I, Akbar H. Chowdhury, do hereby grant permission to the Wallace Library of the Rochester Institute of Technology to reproduce my thesis in whole or in part. Any reproduction shall not be for commercial use or profit.

Date: February 3rd 1998 Author's Signature_____

Rochester Institute of Technology
College of Engineering
Department of Mechanical Engineering

# OFFICE MEMO

To: Professors Ghosh, Gupta and Hennessey
From: Charles W. Haines, M.E. Department Head
Date: 12/30/97
Subject: BS/MS Completion

The following three students are the only remaining BS/MS students who 'graduated' in the Spring of 97. As you know, they should finish their Thesis work for the MS no more than 12 months after finishing their course work - which was either last Spring or Summer. It is my understanding that they are all 'hard' at work - but I would like, from each of you, your best estimate as to when the student working for you will be defending their Thesis.

1) Akbar Chowdhury _____

2) Ron Dufort _____

3) Gary Richardson _____

Thanks for your assistance.

Copy: Fredda Bishop
Akbar Chowdhury ✓
Ron Dufort
Gary Richardson

## ABSTRACT

A number of iterative techniques have recently been developed which are extremely efficient at solving systems of linear equations. Of these methods probably the most recognized is the Conjugate Gradient Method (CG). This is an extremely efficient solver and has been used successfully for a number of years now. A newer method proposed initially by Davidson [1] is studied in this paper. This method has proven itself in terms of efficiency by solving the same system (of order 2000) that was solved by the CG method. It converged in approximately 40 iterations, taking less than five minutes to do so[5], compared to the CG method which took nearly 100 iterations, converging after about 15 minutes. Very little documentation about the derivation or development of Davidson's method exists, and his paper was written in terms of an eigenvalue problem. A portion of a program developed by NASA Ames Research Center uses a variation of Davidson's method as a linear solver. Davidson's method was explored and derived using his paper and the FORTRAN code from NASA. The purpose of this thesis is to provide some insight into the analytical aspect of Davidson's method, using the CG method for comparison.

# Table of Contents


## I. Introduction and Background

(i) The Solution of a Linear System of Equations 1
(ii) The Quadratic Form 3
(iii) Relationship between Eigenvalues and F(**x**) 6
(iv) Relaxation Methods 7
(v) Gradient Methods 12

## II. Conjugate Gradient Method

(i) Description of Conjugate Gradient Process 16
(ii) Derivation and Properties 17
(iii) Graphical Description (based on 2-D case) 18
(iv) Algorithm and Simplifications 22

## III. Interpretation of a Linear Solver based on Davidson's Algorithm

(i) Introductory Comments 26
(ii) Description of Davidson's Process 26
(iii) Derivation and Properties
  (a) Establishing an Orthogonal Residual Vector 30
  (b) Error-Correction of Orthogonal Vector 35
  (c) Convergence Criteria 36
(iv) Basic Algorithm and Computational steps 37
(v) Improving the Procedure & Final Algorithm
  (a) Reducing the Spectral Radius 40
  (b) Condensing the Orthonormal Basis 43
  (c) The Final Algorithm 46
  (d) A Graphical Description of the Process 48

## IV. Conclusion

(i) Illustrative Examples 52
(ii) Concluding Remarks 61

## V. References 63

## VI. Appendix

(i) Matlab programs for Conjugate Gradient and Davidson's Method *i*
(ii) Original FORTRAN source code from NASA Ames Research Center *ix*

# I. Introduction and Background

## (i) The solution of a linear system of equations:

In linear algebra there are two basic problems. The first problem is the solution of a linear system $A\mathbf{x} = \mathbf{b}$, where $\mathbf{x}$ is the solution vector, A is the coefficient matrix and $\mathbf{b}$ is a known constant vecor. The other problem is the eigenvalue problem, which is the solution of the system $A\mathbf{x} = \lambda\mathbf{x}$, where for a particular value of $\lambda$ there is an associated vector $\mathbf{x}$ for which this relationship holds true.

This thesis deals mainly with the solution of a linear system of equations. There are two types of solution methods that are generally seen. One is the direct method, which would use Gaussian elimination to reduce the matrix A to upper triangular form and then use back substitution to produce the solution vector $\mathbf{x}$. The other type of method is the iterative method, of which there are many different types. Some of the older iterative schemes include the Jacobi iteration scheme, the Gauss-Seidel iterative scheme (which is a variation of Jacobi's method), as well as Successive Over Relaxation (SOR) method. All of these methods tend to be slow to converge, and convergence is not always guaranteed.

Given the system of equations:

$$A\mathbf{x} = \mathbf{b},$$

for a guess vector $\mathbf{x}^{(k)}$, the residual vector will have the form

$$\mathbf{r}^{(k)} = \mathbf{b} - A\mathbf{x}^{(k)}, \qquad k = 0,1,2 \ldots n$$

The traditional methods described above will keep on iterating until the norm of the residual vector reduces to below a certain convergence criteria. i.e.,

$$\| \mathbf{r}^{(k)} \| = \| \mathbf{b} - A\mathbf{x}^{(k)} \| \leq \varepsilon \qquad \text{where } \varepsilon \text{ is a small finite number}$$

If we look at multiplication by the matrix A as a transformation matrix which maps a vector $\mathbf{x}^{(k)}$ into the range space of A, creating a new vector $A\mathbf{x}^{(k)}$, then the vector $\mathbf{x}$ which is transformed into the vector $\mathbf{b}$ is the solution of the linear system of equations. Furthermore, when $A\mathbf{x}^{(k)} \neq \mathbf{b}$, the residual vector $\mathbf{r}^{(k)}$ indicates how much the transformed vector $A\mathbf{x}^{(k)}$ needs to change in order for it to coincide with the vector $\mathbf{b}$. The residual vector $\mathbf{r}^{(k)}$ does not give a direct indication of how $\mathbf{x}^{(k)}$ should change so that it lies in the solution space of the system, but it can be used to indicate in an indirect manner how $\mathbf{x}^{(k)}$ can be varied to direct it toward the solution space. When the magnitude of this residual is very small, the vector $A\mathbf{x}^{(k)}$ is essentially the same as the vector $\mathbf{b}$ (see Figure 1), as long as the condition number is not very high. The vector $A\mathbf{x}^{(k)}$ however, will only truly be the same as the vector $\mathbf{b}$ when the residual vector $\mathbf{r}^{(k)}$ is the null vector. Since iterative procedures programmed on computers use floating point arithmetic, the residual vector will rarely be exactly zero.

This is one way to look at the solution of a linear system of equations. In several iterative schemes, $\mathbf{x}$ is varied such that the $k^{th}$ component in $\mathbf{r}$ goes to zero (where $k$ indicates the iteration number). i.e. find $\mathbf{x}^{(k)}$ such that the component $r_k$ is

zero, then search for $\mathbf{x}^{(k+1)}$ that will take the $(k+1)^{th}$ component $r_{k+1}$ to zero. In an n-dimensional Euclidean space, the vector **r** will have n components, each of which will need to approach zero, i.e. the null vector.

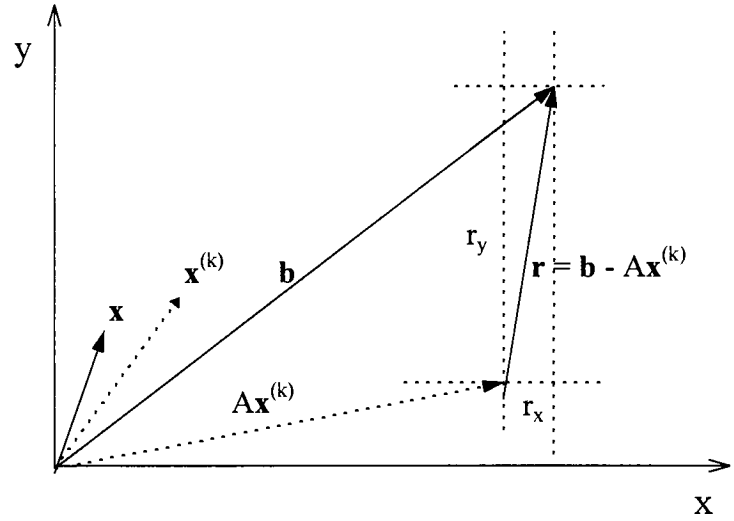



Figure 1: Shows the representation of residual vectors of a system $A\mathbf{x} = \mathbf{b}$. **x** is the true solution, and $\mathbf{x}^{(k)}$ is an approximation.

## (ii) The Quadratic Form:

A symmetric matrix A, which is the coefficient matrix of a linear system of equations, has an associated quadratic form which can be expressed as follows:

$$F(\mathbf{x}) = 1/2\ (\mathbf{x}, A\mathbf{x}) - (\mathbf{b};\mathbf{x}). \qquad (1 - 1)$$

If **x** is an n-dimensional vector, then the quadratic form will have domain $\mathbf{R}^n$.

The graph of F(**x**) will then have (n+1) variables. This can be represented as follows:

$$\{(x_1,\ldots,x_n, x_{n+1}) :(x_1,\ldots, x_n) \in \mathbf{R}^n, x_{n+1} = F(x_1, x_2, \ldots , x_n ), \quad (1 \quad 2)$$

where the expression in 1-2 is essentially the graph of the quadratic function, having domain $\mathbf{R}^{n+1}$. Recall that if we are given a function in two variables, its graph will contain three variables. e.g.

$$x^2 + y^2 = r^2$$

is a circle of radius r in the two dimensional plane. However, when we look at the graph of this function, which has the form:

$$\{(x, y, F(x,y)): F(x, y) = \frac{x^2}{r_a^2} + \frac{y^2}{r_b^2} \}$$

when $r_a \neq r_b$, this will be a paraboloid in $\mathbf{R}^3$. For a symmetric positive definite system (i.e. all the eigenvalues are positive), the graph of F(**x**) will be an elliptic paraboloid when **x** is two dimensional. If we look at a contour plot of the graph of F(**x**) in the xy-plane, we will see concentric ellipses which are projections of the paraboloid. The common center of these ellipses coincides with the minimum of F(**x**). The decreasing sizes of the ellipses in the xy-plane corresponds to the decreasing value of F(x, y), thus tending toward a minimum point. When the contours diminish to a point, it is a minimum of F(x, y). A simple plot is shown to illustrate (Figures 2a and 2b).

Given a system of two equations, the function F(**x**) defined by eqn. 1-1 has a minimum value that corresponds to the solution of A**x** = **b**. The problem can then be reduced to one of finding the location of the minimum value of the quadratic function $F(x_1, x_2)$, or in general $F(x_1, x_2,..., x_n)$.

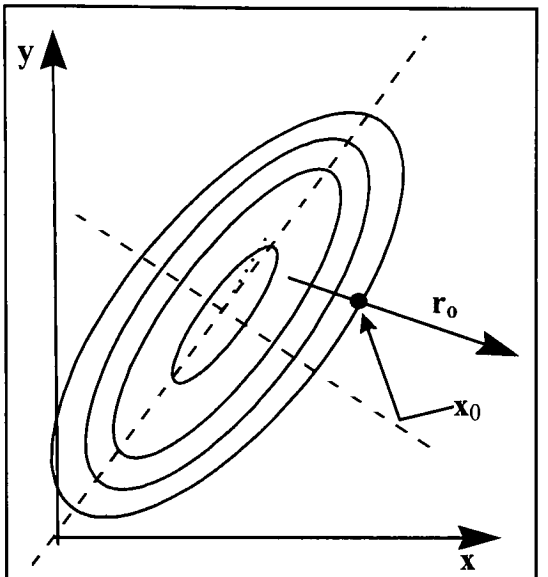



Figure 2a: Contour plot of F(**x**)

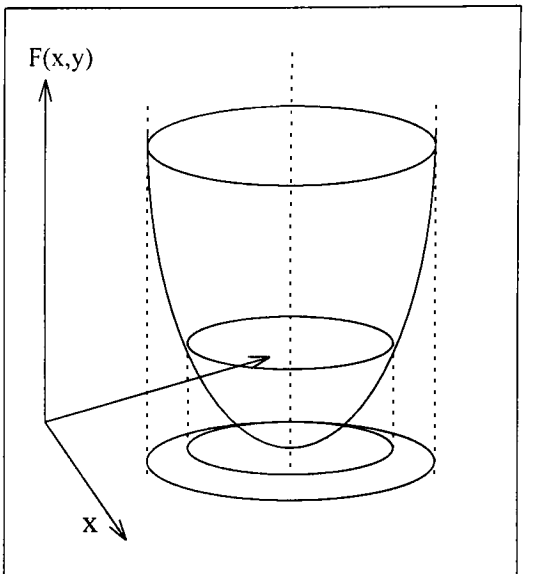



Figure 2b: Graph of two-dimensional Function F(**x**).

If we take the partial derivatives of the quadratic function with respect to each of the components of the vector **x**, we get

$$\frac{\delta F}{\delta x_i} = \sum A_{i\,k}\, x_k - b_i = r_i$$

where $r_i$ is the i $^{th}$ component of the residual vector. This is the gradient of the quadratic function of the linear system. It can also be expressed as:

$$\nabla F(\mathbf{x}^{(k)}) = A\mathbf{x}^{(k)} \quad \mathbf{b} = \mathbf{r}^{(k)} \qquad (1 - 3)$$

where $\mathbf{r}^{(k)}$ is the residual vector corresponding to $\mathbf{x}^{(k)}$. At this point we see another interpretation of the residual vector in relation to the quadratic function F(**x**). The residual vector at the point defined by the position vector $\mathbf{x}^{(k)}$ is also the vector normal to the contour which passes through that point. In Figure 2a, at the point $\mathbf{x}_0$ the residual vector $\mathbf{r}_0$ is normal to the ellipse passing through $\mathbf{x}_0$. This fact is used in the formulation of the Conjugate Gradient method.

### (iii) Relationship between Eigenvalues and F(x):

The curve of the quadratic function F(**x**) of a symmetric positive definite system of equations will be an ellipse when n = 2, an ellipsoid when n = 3, and some other form of an ellipse for n > 3. The eigenvalues of the coefficient matrix of a linear system are related to the axes of the ellipse/ellipsoid described in the previous section. Finding the eigenvalues of a system corresponds to the diagonalization of the matrix A. In other words, it is an elimination of the cross-product terms from the quadratic form of the equation such that its curve appears in standard form, with its center at the origin, and the axes of the curve lying along the coordinate axes of the transformed system. This can also be seen as a rotation of the orthonormal basis corresponding to the original coordinate axes, such that it lies along the orthonormal basis defined by the eigenvectors of the system.

The diagonalized matrix for a system with n = 2 has the quadratic function

$$F(\mathbf{y}) = (\lambda_1 y_1^2 + \lambda_2 y_2^2)/2 - b_1 y_1 - b_2 y_2 ,$$

where $\lambda_1$ and $\lambda_2$ are the eigenvalues of the original matrix. The *y* terms are simply an orthogonal coordinate transformation of the *x* terms, such that they can be represented by a rotation of the coordinate axes. The terms with elements of the vector **b** as their coefficients merely represent a translation of the center of the ellipse, away from the origin of coordinates.

Referring back to the equation of an ellipse in standard form, we know that the denominators of the square terms correspond to the lengths of the semi-major and semi-minor axes of the ellipse. It can therefore be seen that when the equation for $F(\mathbf{y})$ is put into standard form, the semi-major and semi-minor axes will be inversely proportional to the square roots of the minimum and maximum eigenvalues respectively. Furthermore, when the range of eigenvalues ($\lambda_{max} - \lambda_{min}$) is large, this will correspond to a "flattening" of the ellipse, such that use of the norm of the residual vector as a measure of convergence will no longer be accurate. The change in the value of the function $F(\mathbf{x}^{(k)})$ will be small for a relatively large change in the position of the approximation $\mathbf{x}^{(k-1)}$.

### (iv) Relaxation Methods:

Relaxation methods are based on the quadratic forms of linear system of equations, and the fact that the minimization of the quadratic function $F(\mathbf{x})$ is equivalent to the problem of solving a symmetric definite system of equations. A quick proof of this fact is seen from the following proof by example:

From eqn 1-1

$$F(\mathbf{x}^{(k)}) = (\mathbf{x}^{(k)}, A\mathbf{x}^{(k)})/2 - (\mathbf{b}, \mathbf{x}^{(k)}).$$

For a linear system of two equations, $F(\mathbf{x})$ can also be expressed as

$$F(x_1, x_2) = [\, A_{11}x_1^{\,2} + (A_{21} + A_{12})x_1x_2 + A_{22}x_2^{\,2}]/2 - b_1x_1 - b_2x_2$$

and the partial derivatives with respect to $x_1$ and $x_2$ are

$$F_{x1} = A_{11}x_1 + (A_{21} + A_{12})x_2/2 - b_1 \qquad \text{and,}$$

$$F_{x2} = (A_{21} + A_{12})x_1/2 + A_{22}x_2 - b_2$$

Knowing that A is a symmetric matrix, $A_{12} = A_{21}$, then setting the partial derivative expressions equal to zero gives us the following relations:

$$A_{11}x_1 + A_{12}x_2 - b_1 = 0$$

$$A_{21}x_1 + A_{22}x_2 - b_2 = 0$$

which is equivalent to the solution of the linear system of equations $A\mathbf{x} = \mathbf{b}$.

Relaxation methods use a form of the following relation (1-4) to point an approximation toward the true solution. A non-zero direction vector **p** is selected and a scalar multiple of this direction is used to correct the initial guess. The intention is to decrease the value of the quadratic function $F(\mathbf{x})$ at each iteration until the minimum is reached. The general form of the relation that is used is

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + t\mathbf{p}. \qquad (1 - 4)$$

Starting with some $\mathbf{x}^{(k)}$, and having chosen a direction **p** by which to correct this quantity, we see that F is a quadratic function of the scalar multiple t only, since the other variables have been fixed. Substituting eqn 1-4 into eqn 1-1,

$$F(\mathbf{x}^{(k+1)}) = F(\mathbf{x}^{(k)} + t\mathbf{p}) = (A(\mathbf{x}^{(k)} + t\mathbf{p}), (\mathbf{x}^{(k)} + t\mathbf{p}))/2 - (\mathbf{b}, \mathbf{x}^{(k)} + t\mathbf{p})$$

$$= (A\mathbf{x}^{(k)}, \mathbf{x}^{(k)})/2 + t(A\mathbf{x}^{(k)}, \mathbf{p}) + t^2(A\mathbf{p}, \mathbf{p})/2 - (\mathbf{b}, \mathbf{x}^{(k)}) - t(\mathbf{b}, \mathbf{p})$$

giving us

$$F = t^2(A\mathbf{p}, \mathbf{p})/2 + t(\mathbf{r}, \mathbf{p}) + F(\mathbf{x}^{(k)}).$$

For a given direction vector $\mathbf{p}$ and position vector $\mathbf{x}^{(k)}$, it is possible to find a value of the parameter t such that it will locally minimize the function F along the direction of $\mathbf{p}$. This is done by taking the derivative of F with respect to t and then setting the resulting expression equal to zero and solving for t. This gives

$$\frac{dF}{dt} = t(A\mathbf{p}, \mathbf{p}) + (\mathbf{r}, \mathbf{p}) = 0.$$

The expression above gives rise to

$$t_{min} = -\frac{(\mathbf{r}, \mathbf{p})}{(A\mathbf{p}, \mathbf{p})}, \qquad (1 - 5)$$

where, $\mathbf{r} = A\mathbf{x} - \mathbf{b}$.

It can therefore be said that $t_{min}$ is an optimization parameter which tells us how far to travel along a given direction $\mathbf{p}$, such that the function F is locally minimized, and globally decreased as much as possible under the given set of constraints. After correcting $\mathbf{x}^{(k)}$, the process can be repeated iteratively, selecting a new direction $\mathbf{p}$ each time, and optimizing the parameter t until the global minimum of $F(\mathbf{x})$ is reached. Relaxation methods vary according to the manner in which the direction vector $\mathbf{p}$ is chosen, and otherwise use the same procedure outlined above.

Using the optimization parameter $t_{min}$ it is seen that the new residual vector $\mathbf{r}^{(k+1)}$ is orthogonal to the direction vector $\mathbf{p}$ at the new coordinate $\mathbf{x}^{(k+1)}$.

The proof of this is simple:

Using

$$\mathbf{r}^{(k+1)} = A\mathbf{x}^{(k+1)} - \mathbf{b} = A(\mathbf{x}^{(k)} + t\mathbf{p}) - \mathbf{b} = \mathbf{r}^{(k)} + tA\mathbf{p}$$

and taking the inner product on both sides with **p**, we get

$$(\mathbf{r}^{(k+1)}, \mathbf{p}) = (\mathbf{r}^{(k)}, \mathbf{p}) + t(A\mathbf{p}, \mathbf{p}).$$

Setting $t = t_{min}$ (using eqn 1-5 for $t_{min}$) yields

$$(\mathbf{r}^{(k+1)}, \mathbf{p}) = 0.$$

The fact that the residual vector $\mathbf{r}^{(k+1)}$ is orthogonal to the direction vector **p** (also referred to as the relaxation direction) implies that when a relaxation direction is selected, the scalar $t_{min}$ is a search along that direction for the elliptic contour to which the vector **p** is tangent. At the same time, for any given point, the residual vector is normal to the contour that passes through that point. All of this can be seen in Figure 3, where the elliptic contours of a two-dimensional system are sketched along with a starting point $\mathbf{v}^{(0)}$, a new point $\mathbf{v}^{(1)}$, their associated residual vectors, and arbitrary relaxation directions $\mathbf{p}^{(k)}$, subject to the constraint that $\mathbf{p}^{(k+1)}$ is not orthogonal to the residual vector $\mathbf{r}^{(k)}$ from the point $\mathbf{v}^{(k)}$.

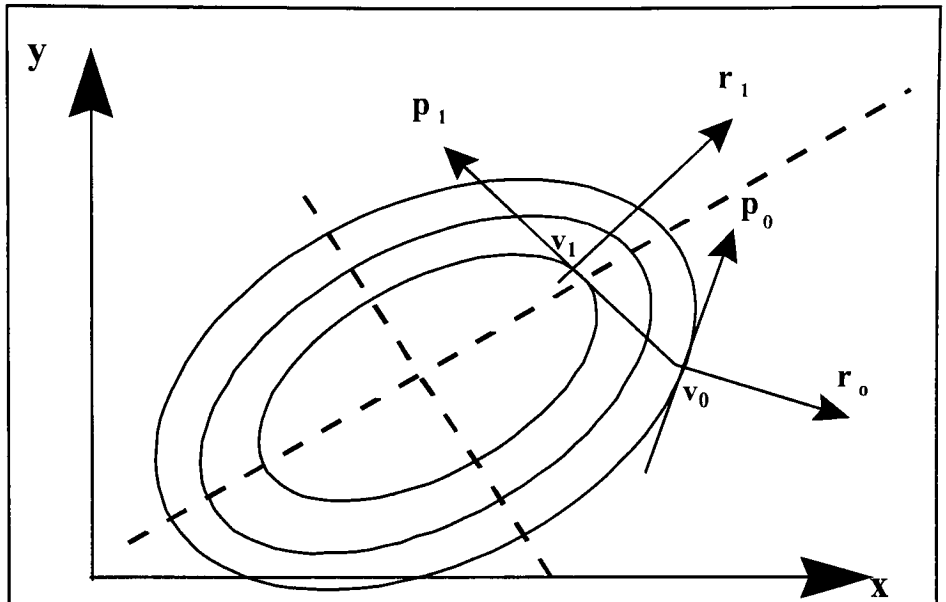


Figure 3: Graphical representation of Relaxation Methods

The individual relaxation methods will not be discussed here. Only the development of the parameter $t_{min}$ was necessary as it will be referred to later in the derivation of the Congjugate Gradient method. However, suffice it to say that the relaxation methods do not use any information about the direction of the residual vector. These methods simply use its norm to check for convergence. On the other hand, both the Conjugate Gradient method as well as Davidson's method make use of the direction of the residual vector to approach the solution. It will be seen later that use of the residual vector leads to a guarantee of convergence within a finite number of steps.

In general, relaxation methods are very slow to converge. The rate can be increased using some variation of the original iteration equation to significantly

increase the rate of convergence relative to other relaxation techniques. For example, use of the parameter $\omega$ in the Successive Over-Relaxation method (SOR) increases the rate of convergence quite significantly in comparison with Jacobi's method, or even the Gauss-Seidel method. However, the convergence rates are still far below those seen in Davidson's method or the Conjugate Gradient method.

### (v) Gradient Methods:

The Gradient methods use a variation of the Relaxation principles. The most general difference between Gradient methods and Relaxation methods is the fact that the Gradient methods utilize the direction of the residual vector to calculate the next relaxation direction. In fact, this is what distinguishes gradient methods from relaxation methods.

The residual vector $\mathbf{r}$, corresponding to the gradient of the function $F(\mathbf{x})$ from a point $\mathbf{x}$, points in the direction that increases the value of $F(\mathbf{x})$ at the maximum local rate of change. Therefore, in order to decrease the function it makes sense to travel in a direction opposite to the direction of $\mathbf{r}$. However, since the maximal rate of increase is only local (hence the rate of decrease will be as well), traveling in a direction opposite to the residual vector is not necessarily the best direction to travel in order to achieve the maximum possible decrease globally.

In the method of Steepest Descent, the relaxation direction is simply the negative of the residual direction, such that we have

$$\mathbf{p}^{(k)} = -\mathbf{r}^{(k-1)} \quad \text{for } k > 1,$$

such that the scalar multiple $t_{min}$ defined above is calculated from

$$t_{min} = \frac{(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)})}{(A\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)})} .$$

As a result of this it is now apparent that in the method of Steepest Descent the $k^{th}$ residual vector is orthogonal to the $(k-1)^{th}$ residual vector. This is true only for two consecutive residual vectors, so that the $k^{th}$ residual vector need not be orthogonal to the $(k-j)^{th}$ residual vector, where $j > 1$. The proof that consecutive residual vectors are orthogonal can be seen by an extension of the proof given in the last section, substituting $\mathbf{r}^{(k)}$ for $\mathbf{p}$. Geometrically, this process follows a piece-wise linear path with right angled corners until the solution is approximated to within the convergence tolerance. This method is generally very slow to converge, and this illustrates how traveling in the optimal local relaxation direction ($-\mathbf{r}$) for the optimal distance along that direction ($t_{min}$), is not sufficient to ensure that a high rate of convergence will be achieved.

The Simultaneous Displacement method uses a fixed parameter t rather than calculating a new value each time, which in itself reduces the computation time. However, the minimum value of $F(\mathbf{x})$ along the direction of the residual vector will not be achieved in each iteration. In order for this method to be successful the value

of t must lie within the range $0 < t < 2/\lambda_{max}$, where $\lambda_{max}$ is the maximum eigenvalue of the matrix A. The following relation is used:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - t\mathbf{r}^{(k-1)}.$$

Before beginning the iteration process, the system of equations $A\mathbf{x} - \mathbf{b} = 0$ can be altered so that the diagonal elements of the matrix A are all unity. This is simply a trivial substitution of variables, and a sequence of elementary row operations which will maintain symmetry while ensuring that the new system of equations

$$A'\mathbf{x} - \mathbf{b}' = 0$$

still has the same solution vector $\mathbf{x}$, and allows the matrix A to be decomposed into the following sum:

$$A' = L + I + U,$$

where L and U are the lower and upper triangular elements of A' respectively. In addition, the value of t can be chosen as unity, and as long as the matrix A has strong diagonal dominance, the value of $\lambda'_{max}$ (the maximum eigenvalue of the matrix A') will generally be less than two, and the criteria for the range of allowable values of t will have been met. The equation correcting $\mathbf{x}^{(k-1)}$ is reduced to

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - \mathbf{r}^{(k-1)} \qquad \text{for } k = 1,2, \ldots$$

for the altered system of equations.

The behavior of this method is different in that it does not follow the relaxation direction to the minimum point defined by eqn (1-5). We know that the

residual vector is defined by

$$\mathbf{r}^{(k-1)} = A'\mathbf{x}^{(k-1)} - \mathbf{b}',$$

and substituting this into the expression above gives us

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} - A'\mathbf{x}^{(k-1)} + \mathbf{b}',$$

where $A' = L + I + U$. It follows that this equation can be written as follows

$$\mathbf{x}^{(k)} = I\mathbf{x}^{(k-1)} - (L + I + U)\mathbf{x}^{(k-1)} + \mathbf{b}',$$

which reduces to

$$\mathbf{x}^{(k)} = -(L + U)\mathbf{x}^{(k-1)} + \mathbf{b}'.$$

The coefficient of $\mathbf{x}^{(k-1)}$ is the iteration matrix for the Simultaneous Displacement method. Which can be designated by $\mathbf{M = -(L + U)}$. It can also be proven that as long as the matrix A is strongly diagonally dominant, the method is guaranteed to converge. A specific relationship between the off-diagonal elements and the diagonal elements of the matrix A can be derived that would ensure convergence, and it would be different for matrices of different sizes. Generally, if the diagonal elements of each of the rows are greater than the sum of the off-diagonal elements of the corresponding rows, the matrix is considered to be strongly diagonally dominant, and should converge using this method.

The simultaneous displacement method tends to be slow to converge. In fact, it is generally much slower than relaxation methods such as the successive displacement method, or the successive over-relaxation method (SOR).

# II. Conjugate Gradient Method:

## (i) Description of Conjugate Gradient Process

The Conjugate Gradient Method goes through an initialization step which is identical to the first step of the method Steepest Descents. A trial vector is selected, and the first relaxation direction is taken as the negative of the corresponding residual vector. The same formula that was used to calculate $t_{min}$ for the relaxation methods is used to determine how far to travel along the direction of the residual vector. The following relation is used for this first step:

$$\mathbf{x}^{(1)} = \mathbf{x}^{(0)} - q_1\mathbf{r}^{(0)}, \qquad (2 - 1)$$

where $q_1$ is merely the parameter t from eqn (1-5), as can be seen in

$$q_1 = \frac{(\mathbf{r}^{(0)}, \mathbf{p}^{(1)})}{(A\mathbf{p}^{(1)}, \mathbf{p}^{(1)})}, \qquad \text{where } \mathbf{p}^{(1)} = -\mathbf{r}^{(0)}. \qquad (2\ \ 2)$$

After completion of this initialization step, the relaxation direction is no longer based solely on the residual vector. The search takes place in the plane spanned by the residual vector $\mathbf{r}^{(1)}$ and the previous relaxation direction $\mathbf{p}^{(1)}$, originating at the point $\mathbf{x}^{(0)}$. For any given iteration the search starts from the most recent approximation to the solution $\mathbf{x}^{(k)}$, and takes place in the plane spanned by the most recent residual vector $\mathbf{r}^{(k-1)}$ and the previous relaxation direction $\mathbf{p}^{(k-1)}$.

This choice of search direction stems from the fact that the two dimensional plane spanned by $\mathbf{r}$ and $\mathbf{p}$ intersects the quadratic function $F(\mathbf{x} = \mathbf{x}^{(k-1)})$ in the form of

an ellipse. Since this ellipse passes through $\mathbf{x}^{(k-1)}$, and is tangent to the relaxation direction $\mathbf{p}^{(k-1)}$ – because the minimization quantity $(q_{(k-1)})$ was used to calculate the location of the new solution vector – $\mathbf{x}^{(k-1)}$ is a minimum point, i.e. it corresponds to the lowest value of $F(\mathbf{x})$ that can be achieved in the search plane spanned by $\mathbf{p}^{(k-2)}$ and $\mathbf{r}^{(k-2)}$. In the new search plane, the center of the ellipse will be the location of the local minimum of the quadratic function $F(\mathbf{x})$ -- in the plane spanned by $\mathbf{r}$ and $\mathbf{p}$.

### (ii) Derivation and Properties

In order for the relaxation direction $\mathbf{p}^{(k)}$ to point from the trial vector $\mathbf{x}^{(k-1)}$ to the center of the ellipse, it is required that the vectors $\mathbf{p}^{(k)}$ and $\mathbf{p}^{(k-1)}$ be conjugate directions with respect to *any* ellipse of intersection (also represented as the intersection of the search plane and $F(\mathbf{x})$ = constant). The following relationship is derived from this property:

$$(A\mathbf{p}^{(k)}, \mathbf{p}^{(k-1)}) = (\mathbf{p}^{(k)}, A\mathbf{p}^{(k-1)}) = 0. \qquad (2-3)$$

By taking a linear combination of the residual vectors and relaxation directions, where unless $\mathbf{x}^{(k)}$ is the solution vector, $\mathbf{r}^{(k)}$ is non-zero, and its value taken as (-1) for the purpose of normalizing, we arrive at the following expression:

$$\mathbf{p}^{(k)} = -\mathbf{r}^{(k-1)} + e_{k-1}\mathbf{p}^{(k-1)}, \qquad \text{for } k = 2, 3, \ldots . \qquad (2-4)$$

Based on the conjugate direction relationship, the coefficient $e_{k-1}$ is calculated from

$$e_{k-1} = \frac{(\mathbf{r}^{(k-1)}, A\mathbf{p}^{(k-1)})}{(\mathbf{p}^{(k-1)}, A\mathbf{p}^{(k-1)})} \quad \text{for } k = 2, 3, \ldots . \qquad (2-5)$$

Having obtained the relaxation direction $\mathbf{p}^{(k)}$, the scalar parameter that will minimize $F(\mathbf{x})$ can be calculated to determine how far to travel along the new relaxation direction. Using $q_k$ in place of $t_{min}$ we now have the following expression:

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + q_k \mathbf{p}^{(k)}$$

where

$$q_k = - \frac{(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k)})}{(A\mathbf{p}^{(k)}, \mathbf{p}^{(k)})} \qquad \text{for } k = 2, 3, \ldots.$$

As a result of the positive definiteness of the matrix A, the denominators in the expressions for $q_1$, $e_{k-1}$, and $q_k$ will be positive as long as the relaxation direction $\mathbf{p}^{(k)}$ is non-zero.

After the initialization step described above, the iterative steps consist of calculating the new relaxation direction $\mathbf{p}^{(k)}$, and then calculating the new approximation to the solution vector $\mathbf{x}^{(k)}$, using the formulas listed above. This cycle is repeated until convergence is achieved to within the desired tolerance.

### (iii) Graphical Description (based on two-dimensional case)

The graphical interpretation of the conjugate gradient will now be stated with reference to Figure 4. The residual vectors and relaxation vectors have been underlined to indicate that they represent directions, while the labels that are not underlined represent position vectors only. Starting at some arbitrary initial guess $\mathbf{x}_0$,

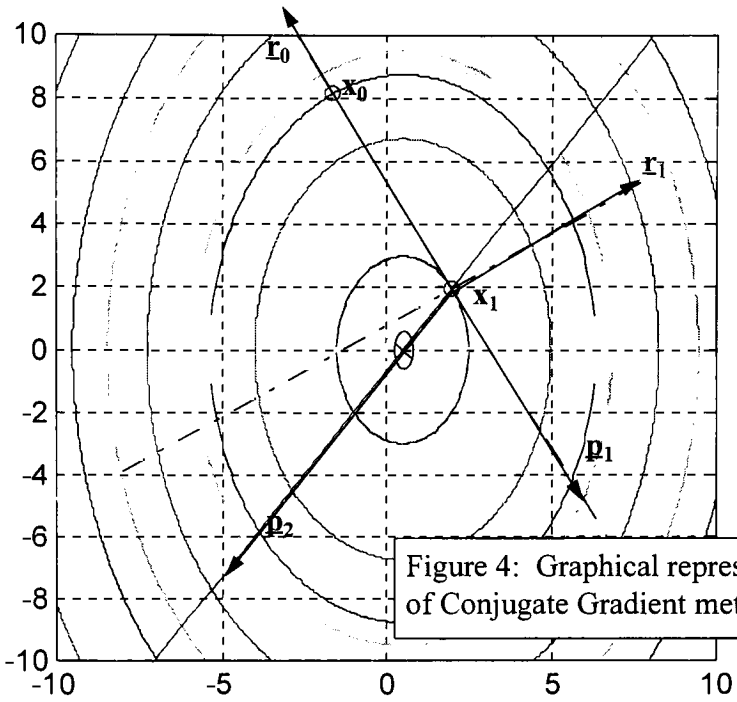



Figure 4: Graphical representation of Conjugate Gradient method.

the residual vector $\mathbf{r}_0$ is calculated, and it is the outward pointing normal of the ellipse that passes through $\mathbf{x}_0$. Since we know that the vector normal to an ellipse points away from the center of the ellipse, it makes sense to travel in a direction opposite to it. This is the direction $\mathbf{p}_1$, and it is the initial relaxation direction of the Conjugate Gradient method. If we travel a distance $q_1$ along the relaxation direction we arrive at the point $\mathbf{x}_1$. It can be seen that $\mathbf{p}_1$ is tangent to the ellipse that passes through $\mathbf{x}_1$. At this point, the vector $\mathbf{r}_1$ is orthogonal to $\mathbf{p}_1$ and is the outward normal of the ellipse passing through $\mathbf{x}_1$ at that point. The iteration process begins at this point and the new relaxation direction lies in the plane spanned by the vectors $\mathbf{r}_1$ and $\mathbf{p}_1$.

A property of the relaxation directions that was mentioned before is the fact that they are conjugate directions. Another interpretation of this is that if we are given an ellipse along with the coordinates of one of the points lying on that ellipse, and the vectors tangent to the ellipse and normal to it at the given coordinate are known, then the product of the coefficient matrix A – which defines the profile of the ellipse – and the vector tangent to the ellipse will produce a vector which points from the given coordinate to the center of the ellipse which lies on the two-dimensional plane spanned by the normal and tangent vectors. This statement applies to Euclidean Spaces of all sizes ($\mathbf{R}^n$).

Given the ellipse of intersection, it is quite straight-forward to show that the conjugate direction requirement on the relaxation directions in that plane, forces the trial vector to point toward the center of the ellipse. The proof is as follows:

> Let us consider the two-dimensional case for simplicity. The first relaxation direction is the negative of the residual direction, and is tangent to the ellipse of intersection, with the second residual normal to it.
>
> The residual vector of the first iteration has the form
>
> $$\mathbf{p}^{(1)} = -\mathbf{r}^{(0)} = -A\mathbf{x}^{(0)} + \mathbf{b}.$$
>
> If we take the second relaxation direction as the vector pointing

from the second trial solution $\mathbf{x}^{(1)}$ to the center of the ellipse (i.e. the solution of the two-dimensional system, $\underline{\mathbf{x}}$),

$$\mathbf{p}^{(2)} = \underline{\mathbf{x}} - \mathbf{x}^{(1)} = A^{-1}\mathbf{b} - \mathbf{x}^{(1)}$$

then the inner product $(\mathbf{p}^{(1)}, A\mathbf{p}^{(2)})$ will be

$$\begin{aligned}(\mathbf{p}^{(1)}, A\mathbf{p}^{(2)}) &= (-A\mathbf{x}^{(0)} + \mathbf{b}, A(A^{-1}\mathbf{b} - \mathbf{x}^{(1)})) \\ &= (-A\mathbf{x}^{(0)}, \mathbf{b} - A\mathbf{x}^{(1)}) + (\mathbf{b}, \mathbf{b} - A\mathbf{x}^{(1)}) \\ &= (-A\mathbf{x}^{(0)}, \mathbf{r}^{(1)}) + (\mathbf{b}, \mathbf{r}^{(1)}) \\ &= (\mathbf{r}^{(1)}, -A\mathbf{x}^{(0)}+\mathbf{b}) \\ &= (\mathbf{r}^{(1)}, \mathbf{r}^{(0)}) = 0\end{aligned}$$

It is known that the residual vectors from any two different iterations will be orthogonal, therefore their inner product will be zero. We see here that in order for the new relaxation direction $\mathbf{p}^{(k)}$ to point toward the center of the ellipse, the vectors $\mathbf{p}^{(k)}$ and $\mathbf{p}^{(k-1)}$ must be conjugate directions as defined above. The proof for the case of $(\mathbf{p}^{(2)}, A\mathbf{p}^{(1)})$ is similar.

We can also look at the Conjugate Gradient method in another way as a result of this proof. Since the method seeks the center of the ellipse in a two dimensional plane for each iteration, it is essentially breaking the problem of solving an n-dimensional system of linear equations into a series of two dimensional problems which are solved in sequence until the solution is reached. This does not mean that it is merely a series of equations where the coefficient matrix A is a 2x2 matrix. Rather,

the search for the solution takes place in a two-dimensional subspace spanned by two orthogonal vectors, and each of these planes is orthogonal to the previous one.

### (iv) Algorithm and Simplifications

The following algorithm was used in Matlab to program the Conjugate Gradient method. The algorithm is taken from a text by Schwarz[2].

After an arbitrary initial guess at the solution, the following steps are carried out:

$$\mathbf{r}^{(0)} = A\mathbf{x}^{(0)} + \mathbf{b}, \qquad \text{and} \qquad \mathbf{p}^{(1)} = -\mathbf{r}^{(0)}.$$

The iteration sequence can now be begun for k = 1, 2, …

$$e_{k-1} = \frac{(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)})}{(\mathbf{r}^{(k-2)}, \mathbf{r}^{(k-1)})} \qquad k \geq 2 \qquad (2-6)$$

$$\mathbf{p}^{(k)} = -\mathbf{r}^{(k-1)} + e_{k-1}\mathbf{p}^{(k-1)} \qquad (2-7)$$

$$q_k = \frac{(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)})}{(A\mathbf{p}^{(k)}, \mathbf{p}^{(k)})} \qquad (2-8)$$

$$\mathbf{x}^{(k)} = \mathbf{x}^{(k-1)} + q_k \mathbf{p}^{(k)}, \qquad (2-9)$$

$$\mathbf{r}^{(k)} = \mathbf{r}^{(k-1)} + q_k (A\mathbf{p}^{(k)}) \qquad (2-10)$$

The expressions listed above are not the same as the ones that were shown at the beginning of subsection **(ii)**. A number of simplifications were used to arrive at these new expressions. They are essentially equivalent to the original equations

shown at the beginning of subsection **(ii)**. The requirement that consecutive relaxation vectors must be conjugate directions gives rise to a number of simplifications.

The relation for the calculation of the residual vector,

$$\mathbf{r}^{(k)} = A\mathbf{x}^{(k)} - \mathbf{b} = A\mathbf{x}^{(k-1)} + q_k A\mathbf{p}^{(k)} - \mathbf{b} = \mathbf{r}^{(k-1)} + q_k(A\mathbf{p}^{(k)}), \qquad (2-11)$$

is a recursion formula that allows the residual to be calculated using the quantity $A\mathbf{p}^{(k)}$, which must be calculated anyway. Thus, only a vector addition is required at each iteration step rather than a matrix multiplication, which saves on the computation time required. The residual vector $\mathbf{r}^{(k)}$ will be orthogonal to the plane spanned by $\mathbf{r}^{(k-1)}$ and $\mathbf{p}^{(k-1)}$ since $\mathbf{x}^{(k)}$ is the minimum value of $F(\mathbf{x})$ in the plane. This is a result of the fact that at the minimum point, the new residual vector is orthogonal to the relaxation directions. From this observation, the following generalizations are made:

$$(\mathbf{r}^{(k)}, \mathbf{r}^{(k-1)}) = 0$$

$$(\mathbf{r}^{(k)}, \mathbf{p}^{(k-1)}) = 0$$

and $$(\mathbf{r}^{(k)}, \mathbf{p}^{(k)}) = 0.$$

Using the expression for the relaxation direction $\mathbf{p}^{(k)}$ we have

$$\mathbf{p}^{(k)} = -\mathbf{r}^{(k-1)} + e_{k-1}\mathbf{p}^{(k-1)},$$

and if we take the inner product of this expression with $\mathbf{r}^{(k-1)}$,

$$(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k)}) = -(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)}) + e_{k-1}(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k-1)}).$$

Since the it is known that $(\mathbf{r}^{(k)}, \mathbf{p}^{(k)}) = 0$, the same will be true for $(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k-1)})$, such that the term $e_{k-1}(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k-1)})$ from the expression above will drop out, and the following relation will hold true:

$$(\mathbf{r}^{(k-1)}, \mathbf{p}^{(k)}) = -(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)}),$$

giving us the new expression above for $q_k$. Rearranging the recursive formula for the residual vector we get

$$A\mathbf{p}^{(k-1)} = \frac{1}{q_{k-1}}(\mathbf{r}^{(k-1)} - \mathbf{r}^{(k-2)}).$$

Taking the inner product of this expression with $\mathbf{r}^{(k-1)}$ gives

$$(\mathbf{r}^{(k-1)}, A\mathbf{p}^{(k-1)}) = \frac{1}{q_{k-1}}[(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)}) - (\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-2)})] = \frac{1}{q_{k-1}}(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)})$$

This expression allows us to rewrite the expression for $e_{k-1}$ as

$$e_{k-1} = \frac{(\mathbf{r}^{(k-1)}, \mathbf{r}^{(k-1)})}{(\mathbf{r}^{(k-2)}, \mathbf{r}^{(k-2)})} \qquad \text{for } k = 2, 3, \ldots$$

In this method, the residual vectors $\mathbf{r}^{(k)}$ form a system of orthogonal vectors. This was already seen before. At the same time, however, the relaxation directions $\mathbf{p}^{(k)}$ form a system of conjugate directions. The fact that the residual vectors form an orthogonal basis leads us to conclude that the system must theoretically converge within n iterations where *n* is the order of the system. For a given system of linear equations, an orthogonal subspace which spans the entire vector space of the system can have no more than *n* orthogonal vectors. If there are more than *n* vectors then there will be linear dependence among two or more of the vectors in that basis. As a

result, after *n* iterations, there will be *n* orthogonal residual vectors. The only vector which will be orthogonal to the basis containing *n* orthogonal vectors will be the null vector. When the null vector is achieved as the residual vector, the solution is obtained.

It must be pointed out that this observation is only theoretically valid. Since iterative procedures use floating point arithmetic, there will be some error introduced as a result of round-off. Consequently, in practice, convergence may not be achieved within n iterations. One of the reasons for this is that the round-off error introduced leads to a situation where the residual vectors may not be perfectly mutually orthogonal. This means that the Euclidean inner product of any two residual vectors $\{(\mathbf{r}^{(k)}, \mathbf{r}^{(l)}), k \neq l\}$ will not equal zero. The solution to this setback is simple. The iterations are carried out to beyond the $n^{th}$ iteration until satisfactory convergence has been achieved. Since cycling through the process calculates a new relaxation direction, as long as it is non-zero, iteration can be continued. A further discussion of the Conjugate Gradient method for the solution of the eigenvalue problem can be found in Hestenes and Karush[4]

# III. Interpretation of a Linear Solver based on Davidson's Method:

## (i) Introductory comments

The only documentation available regarding this method was Davidson's paper and a FORTRAN subroutine extracted from a program. Davidson's paper involves the solution of an eigenvalue problem and contains only the algorithm for solving the eigenvalue problem. Most methods for obtaining eigenvalues and eigenvectors can be converted into a linear solver. The subroutine from the PMARC program written at the NASA Ames Research Center uses a linear solver based on Davidson's method.

No documentation was available on the development or derivation of the linear solver based on Davidson's method. Using the extracted code, an interpretation of Davidson's method was developed. Based on this interpretation a derivation is presented, which leads to an extremely efficient algorithm for the solution of a linear system of equations.

## (ii) Description of Davidson's Process

Davidson's method is a fairly new iterative scheme that is based on Lanczos Method[3]. It has been observed that this method consistently outperforms the Conjugate Gradient Method. Since the Conjugate Gradient Method is used much

more widely than Lanczos method, this thesis concentrates on the former. Little is said about Lanczos Method, and the Conjugate Gradient method is used to illustrate the significant difference between the efficiencies of the two systems. The Conjugate Gradient method has a number of limitations, almost all of which are overcome in Davidson's Method. The limitations are discussed, and a general description of how Davidson's Method overcomes them is given in this section.

First, the process is not limited to symmetric matrices. In the cases of both CG and Lanczos methods, numerous additional computations must be carried out in order to condition the matrix so that the system can be solved by these processes. This is not the case for Davidson's method. In examples given later, symmetric matrices will be used to illustrate the differences in the performance of the two schemes. Since no additional computations are required for Davidson's method to successfully converge to the solution, it can be concluded that the system will have greater efficiency in solving linear systems of equations even when the coefficient matrices are asymmetric. An example of such a case will be given.

The limitation outlined above can be viewed as a minor set-back. The linear system of equations $A\mathbf{x} = \mathbf{b}$, where the coefficient matrix A is asymmetric, can be transformed into a symmetric system of equations by the following multiplication:

$$A^T A\,\mathbf{x} = A^T\,\mathbf{b},$$

and it will still have the same solution vector $\mathbf{x}$. This procedure adds $(n^3 + n^2)$ computations to the entire procedure, where A is a square matrix whose size is $n$.

However, even after this transformation, the Conjugate Gradient method requires more iterations to converge than Davidson's method. In addition, the original system of equations required storing only the matrix A and the vector **b** in order to define the system. Now however, storage space is required for both the matrix A as well as the matrix $A^TA$. This can be a major obstacle when dealing with large systems of equations.

The major advantage of Davidson's method over the Conjugate Gradient method lies in the manner in which Davidson's method searches for the solution. As stated earlier, the Conjugate Gradient method conducts its search in a two-dimensional plane spanned by two orthogonal vectors. After the plane has been defined, the Conjugate Gradient method searches for the coordinate vector lying in that plane which minimizes the quadratic function $F(\mathbf{x})$. It continues to do this at each iteration, defining a new plane and searching for the minimizing coordinate in that plane each time, until the convergence criteria has been satisfied (i.e. $\| \mathbf{r}^{(k)} \| \leq \varepsilon$, where $\varepsilon << 1$). The next logical question to ask is whether this search can be conducted in a space spanned by more than two vectors. In other words, instead of searching in a space spanned by $\mathbf{R}^2 \subseteq \mathbf{R}^n$ at each iteration, is it possible to search in a space spanned by $\mathbf{R}^k \subseteq \mathbf{R}^n$, where $k \leq n$ ? This is in fact, exactly what Davidson's method does.

Each iteration is a search for the solution vector in an orthonormal base, the size of which increases with each iteration. For example, at the second iteration, the

search is conducted in $\mathbf{R}^2$, and at the $k^{th}$ iteration, the search is being conducted in a space spanned by $\mathbf{R}^k$. The starting search direction is given by the direction of the initial guess. The process searches for the best approximation it can find *along* the direction of the initial guess vector, $\mathbf{v}^{(0)}$. In the next iteration it will search for the solution in the plane spanned by $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$, where the vectors $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$ constitute an orthonormal base spanning $\mathbf{R}^2$, and the following iteration will be conducted in an orthonormal base spanning $\mathbf{R}^3$. The vectors $\mathbf{v}^{(k)}$ are used to denote the orthonormal unit vectors spanning $\mathbf{R}^k$. In the Conjugate Gradient method, the initial search is conducted along the direction of the residual vector $(A\mathbf{x}^{(0)}-\mathbf{b})$ defined by the initial guess $\mathbf{x}^{(0)}$. Subsequent iterations are conducted in planes spanned by the residual vector of the current iteration and the relaxation direction of the previous iteration (conjugate direction from previous iteration). This represents one of the key differences between the two methods.

Another major difference is the use of the residual vector to influence the direction of the search. The Conjugate Gradient method uses the residual vector at each iteration to define a component of the search direction. Davidson's method on the other hand, searches along the orthonormal directions for the approximation to the solution which will make the projection of the residual vector onto the orthonormal base equal zero (refer to figure 5).

## (iii) Derivation and Properties:

### *(a) Establishing an orthogonal residual vector*

Given an arbitrary search direction, let the unit vector pointing in that direction be represented as $\mathbf{v}^{(0)}$. The residual vector $\mathbf{r}_v^{(0)}$ points from the tip of the vector $A\mathbf{v}^{(0)}$ to the tip of the vector **b**. A scalar multiple $\alpha$ of the vector $\mathbf{v}^{(0)}$ is desired such that the residual vector $\mathbf{r}^{(0)}$ will be orthogonal to the direction $\mathbf{v}^{(0)}$, making the projection of the residual $\mathbf{r}^{(0)}$ onto the unit vector direction $\mathbf{v}^{(0)}$ equal to zero (figure 5).

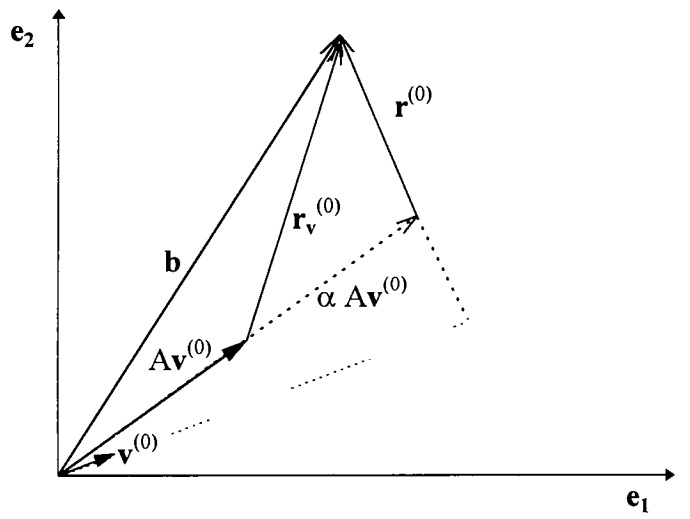



Figure 5: Davidson's method for 2-D case

*Proof of Orthogonality:*

The following simple derivation shows how this can be accomplished:

The residual vector $\mathbf{r}_v^{(0)}$ can be defined by the following expression

$$\mathbf{r}_v^{(0)} = A\mathbf{x} - A\mathbf{v}^{(0)} = \mathbf{b} - A\mathbf{v}^{(0)}.$$

A scalar multiple $\alpha$ of $\mathbf{v}^{(0)}$ is desired such that the new residual vector

$$\mathbf{r}^{(0)} = A(\mathbf{x} - \alpha\mathbf{v}^{(0)}) = \mathbf{b} - \alpha A\mathbf{v}^{(0)}$$

is orthogonal to $\mathbf{v}^{(0)}$. By taking the Euclidean inner product of $\mathbf{r}^{(0)}$ with $\mathbf{v}^{(0)}$, and setting it equal to zero, we get the following

$$(\mathbf{r}^{(0)}, \mathbf{v}^{(0)}) = (\{\mathbf{b} - \alpha A\mathbf{v}^{(0)}\}, \mathbf{v}^{(0)}) = 0. \qquad (3 - 1)$$

By expanding the expression and solving for $\alpha$,

$$(\mathbf{v}^{(0)}, \alpha A\mathbf{v}^{(0)}) = (\mathbf{v}^{(0)}, \mathbf{b})$$

we get

$$\alpha = \frac{(\mathbf{b}, \mathbf{v}^{(0)})}{(\mathbf{v}^{(0)}, A\mathbf{v}^{(0)})}. \qquad (3 - 2)$$

The vector $\alpha\mathbf{v}^{(0)}$ will be the first approximation to the solution vector $\mathbf{x}$. Since the inner product $(\mathbf{r}^{(0)}, \mathbf{v}^{(0)}) = 0$, we know that the residual vector will be orthogonal to $\mathbf{v}^{(0)}$, i.e. the projection of the residual onto the unit vector direction will be zero. Another way to state this is that the residual vector will have a zero component in the direction of the vector $\mathbf{v}^{(0)}$.

After the initial scalar $\alpha$ has been obtained, the next step is to find a new search direction. Since it is known that the residual vector $\mathbf{r}^{(0)}$ is orthogonal to the direction $\mathbf{v}^{(0)}$, a natural choice for the new search direction would be the direction of the residual vector. This is where the true strength of Davidson's method begins to become apparent.

In the Conjugate Gradient method, a direction is chosen, and then the distance to travel along that direction is determined. In Davidson's method, a new search direction is chosen $\mathbf{v}^{(k)}$ (as part of an increasing orthonormal base), and the components of the new coordinate vector are determined simultaneously. This means that the new approximation to the solution $\mathbf{x}^{(k)}$ need not be orthogonal to the previous approximation $\mathbf{x}^{(k-1)}$, and the approximation to the solution at the $k^{th}$ iteration has the freedom of moving about anywhere in $\mathbf{R}^k$ – as opposed to moving in a two dimensional plane at each iteration. This amazing feat is accomplished by an extension of the Proof of Orthogonality:

> Now that there are two orthonormal search directions, two scalar multiples $\alpha_1$ and $\alpha_2$ are desired that would make the residual vector orthogonal to *both* search directions simultaneously. First, let us consider the expression for the approximation to the solution
>
> $$\mathbf{x}^{(2)} = \alpha_1{}^{(2)}\mathbf{v}^{(0)} + \alpha_2{}^{(2)}\mathbf{v}^{(1)}$$

where $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$ are orthonormal directions, and the $\alpha$-terms are the components in those directions which would make the projection of the residual vector onto both direction vectors equal to zero. i.e. we want

$$(\mathbf{r}^{(1)}, \mathbf{v}^{(0)}) = (\mathbf{r}^{(1)}, \mathbf{v}^{(1)}) = 0,$$

giving rise to the following two equations:

$$(\mathbf{b} - A\mathbf{x}^{(1)}, \mathbf{v}^{(0)}) = (\{\mathbf{b} - A(\alpha_1^{(2)}\mathbf{v}^{(0)} + \alpha_2^{(2)}\mathbf{v}^{(1)})\}, \mathbf{v}^{(0)}) = 0 \text{ and,}$$

$$(\mathbf{b} - A\mathbf{x}^{(1)}, \mathbf{v}^{(1)}) = (\{\mathbf{b} - A(\alpha_1^{(2)}\mathbf{v}^{(0)} + \alpha_2^{(2)}\mathbf{v}^{(1)})\}, \mathbf{v}^{(1)}) = 0 .$$

Expanding these two equations would give us a pair of simultaneous equations where it would be necessary to solve for the scalar $\alpha$-terms. The general form of these equations can be expressed as follows:

$$(\{\Sigma_{i=1,k}\, \alpha_i^{(k)} A\mathbf{v}^{(i-1)}\}, \mathbf{v}^{(j)}) = (\mathbf{b}, \mathbf{v}^{(j)}) \text{ for } j = 0, k\text{-}1, \quad (3 - 3)$$

where $k$ is the current iteration number, and there will be a total of $j = k$ equations. This system of equations must to be solved to find the values of $\alpha$. Any direct solver would be sufficient to solve the system of equations, regardless of the size of the original linear system, such as a simple Gauss-Jordan elimination scheme. For iteration $k = 2$, the matrix to be solved would have the following form:

$$\begin{bmatrix} (\mathbf{v}^{(0)}, A\mathbf{v}^{(0)}) & (\mathbf{v}^{(0)}, A\mathbf{v}^{(1)}) \\ (\mathbf{v}^{(1)}, A\mathbf{v}^{(0)}) & (\mathbf{v}^{(1)}, A\mathbf{v}^{(1)}) \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix}^{(k)} = \begin{bmatrix} (\mathbf{v}^{(0)}, \mathbf{b}) \\ (\mathbf{v}^{(1)}, \mathbf{b}) \end{bmatrix} \qquad (3 - 4)$$

Let the coefficient matrix of this equation be designated as W, the vector containing the scalar multiples $\alpha$, and the constant vector **B**. Now we have

$$W\,\alpha^{(k)} = \mathbf{B}^{(k)} \qquad \text{where}$$

$$W = [V^T][A][V\,] \quad \& \quad \mathbf{B} = [V^T][\,\mathbf{b}\,].$$

It can be seen that the size of the coefficient matrix W, the solution vector $\alpha$,and the constant vector **B** increase with each iteration. Notice that the component $W_{(1,\,1)}$ of the coefficient matrix is the same as the corresponding component for the first iteration. This is an interesting fact to keep in mind because all the components of the coefficient matrix W do not need to be calculated. Only the components of the $k^{th}$ row and column need to be calculated for each new iteration, along with the $k^{th}$ component of the constant vector **B**. After the components of this system have been calculated for the current iteration, we solve for the components of the vector $\alpha^{(k)}$.

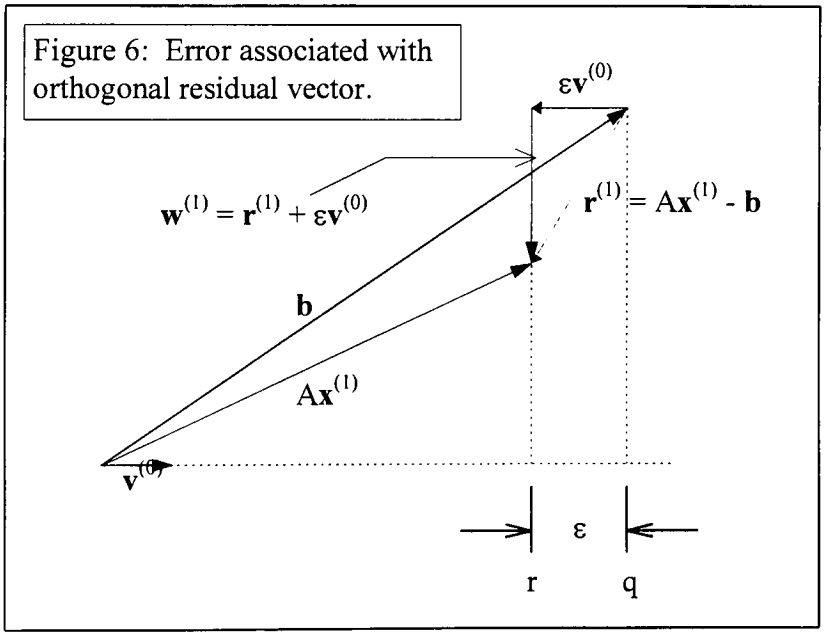
The construction of the components of the system above can be viewed as a series of matrix multiplication. Let the vectors comprising the orthonormal base up to and including the current iteration, be assembled as the column vectors of a matrix V whose size will be (n, k). The linear system above can then be reduced to

(3 - 5) $$[\,V^T\,]_{(k\,,n)}\,[\,A\,]_{(n,\,n)}\,[\,V\,]_{(n,\,k)}\,[\alpha^{(k)}]_{(k,\,1)} = [\,V^T\,]_{(k,\,n)}\,[\,\mathbf{b}\,]_{(n,\,1)}\,.$$

*(b) Error-Correction of orthogonal vector*

Due to the fact that floating point arithmetic is used for these computations, the residual will rarely, if ever, truly be orthogonal to the orthonormal base. Thus, the use of an orthonormalization scheme such as the Gram-Schmidt algorithm can be utilized to improve the orthogonality of the new search direction. Figure 6 shows how such a procedure would help the search direction. The figure is greatly exaggerated for illustration purposes.

Suppose that the projection of the first approximation to the solution $A\mathbf{x}^{(1)}$ reaches point $r$ along the direction $\mathbf{v}^{(0)}$, while the projection of $\mathbf{b}$ reaches the point $q$. The difference between the two projections is the quantity $\varepsilon$, which indicates how much the direction of the residual vector must be altered along $\mathbf{v}^{(0)}$, so that the new



search direction $\mathbf{w}^{(1)}$ is orthogonal to it. In other words, the difference between the two vectors will be the component of the residual vector $\mathbf{r}^{(1)}$ that is orthogonal to the direction $\mathbf{v}^{(0)}$.The quantity $\varepsilon$ can be calculated from

$$\varepsilon = (\mathbf{v}^{(0)}, A\mathbf{x}^{(1)}) - (\mathbf{v}^{(0)}, \mathbf{b}),$$

so that the new search direction has the form

$$\mathbf{w}^{(1)} = \mathbf{r}^{(1)} + \varepsilon\mathbf{v}^{(0)} = \mathbf{r}^{(1)} + (\mathbf{v}^{(0)}, A\mathbf{x}^{(1)} - \mathbf{b})\mathbf{v}^{(0)}$$

$$\mathbf{w}^{(1)} = \mathbf{r}^{(1)} + (\mathbf{v}^{(0)}, \mathbf{r}^{(1)})\,\mathbf{v}^{(0)}. \qquad (3 - 6)$$

The vector $\mathbf{w}^{(1)}$ is the new orthogonal search direction, and after it is normalized, the new orthonormal search direction $\mathbf{v}^{(1)}$ is obtained. The expression for $\mathbf{w}^{(1)}$ is essentially a form of the Gram-Schmidt process. Therefore, it can be extended to any finite dimensional space so that given any vector (residual vector $\mathbf{r}^{(k)}$ in this case), a new vector can be found that is orthogonal to the space spanned by the orthonormal basis $\{\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(k-1)}\}$, producing the orthonormal vector $\mathbf{v}^{(k)}$. It should be kept in mind that the new direction vector is normalized as soon as it is obtained.

*(c) Convergence Criteria*

After each iteration, convergence can be checked in any number of ways. One of the methods that is often used for checking convergence is to calculate the norm of the residual vector, and if it is less than the set tolerance then the process terminates. Another method which can be used is to use the square of the norm, and this choice of

checking convergence will be less strict than the previous as long as $\| \mathbf{r} \| < 1$. The linear solver based on Davidson's method uses another technique of checking convergence, which is to check whether the absolute value of the largest component of a form of the residual vector satisfies the convergence criteria. The following formula is used:

$$\max |( Ax_i - b_i)/b_i| < \varepsilon \qquad \text{for } i = 1, 2, \ldots, n$$

$$= \max |( r_i / b_i)| < \varepsilon \qquad (3 - 7)$$

This is a good enough check of whether the convergence criteria has been satisfied. At the same time, it involves fewer computations than calculating the norm. This convergence check can generally be placed into the following inequality

$$\| \mathbf{r} \|^2 < \max| (r_i / b_i) | < \| \mathbf{r} \| \qquad \text{for } \| \mathbf{r} \| < 1.$$

So far, the general procedure for solving a system of linear equations using Davidson's method has been outlined.

### (iv) Basic Algorithm and Computational Steps

The following computational procedure shows the basic steps of Davidson's method in a compact form. *This is not the final algorithm.*

The variable **y** is a single dimensional buffer array storing n components.

1. Obtain an initial guess $\mathbf{x}^{(0)}$.
2. Normalize $\mathbf{x}^{(0)}$ to obtain $\mathbf{v}^{(0)}$ as the first search direction. The span of $V^{(1)}$ consists of the vector $\mathbf{v}^{(0)}$.

3. The vectors $\{\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots \mathbf{v}^{(k-1)}\}$ are stored in the matrix $V^{(k)}$.
4. Calculate and store the vectors $A\mathbf{v}^{(k)}$ as the columns of a matrix M, $A\mathbf{v}^{(0)}$ is stored in $M^{(1)}$ in this case.
5. Calculate the components of the matrix $W^{(k)}$ and the vector $\mathbf{B}^{(k)}$ from

   $W^{(k)} = [V^T]^{(k)}[\,A\,][V\,]^{(k)} = [V^T]^{(k)}\,[M]^{(k)}$ $\quad (W^{(1)} = [\mathbf{v}^{(0)}\,]^T\,[A\mathbf{v}^{(0)}])$&

   $\mathbf{B}^{(k)} = [V^T]^{(k)}[\,\mathbf{b}\,]$ $\quad (\mathbf{B}^{(1)} = [\mathbf{v}^{(0)}]^T[\,\mathbf{b}\,]).$
6. Use a direct solver such as Gauss-Jordan elimination to solve the following system of equations for $\alpha^{(k)}$

   $W^{(k)}\,\alpha^{(k)} = \mathbf{B}^{(k)}$ $\quad ([W^{(1)}][\alpha^{(1)}] = [\mathbf{B}^{(1)}]).$
7. Calculate the latest approximation to the constant vector using

   $A\mathbf{x}^{(k)} = [M^{(k)}][\alpha^{(k)}]$ $\quad (A\mathbf{x}^{(1)} = [M^{(1)}][\alpha^{(k)}]\,).$
8. Check whether the convergence criteria has been satisfied using

   $\max |(\,Ax_i^{(k)} - b_i)/b_i| < \varepsilon$ $\quad$ for i = 1, 2, …, n

   where $\varepsilon$ is the convergence criteria.
9. Form the new residual vector direction $\mathbf{r}^{(k)}$ from

   $\mathbf{y} = A\mathbf{x}^{(k)} - \mathbf{b}$ $\quad (\mathbf{y} = A\mathbf{x}^{(1)} \quad \mathbf{b}\,)$

   $\mathbf{r}^{(k)} = \mathbf{y} \,/\, \|\,\mathbf{y}\,\|$ $\quad (\mathbf{r}^{(1)} = \mathbf{y} \,/\, \|\,\mathbf{y}\,\|\,)$
10. Calculate the vector $\mathbf{v}^{(k)}$ $(\mathbf{v}^{(1)})$ that is orthogonal to the base $V^{(k)}$ $(V^{(1)})$ using the following simple algorithm for the Gram-Schmidt process:

    $\mathbf{y} = \mathbf{r}^{(k)}$

    for i = 1, k

    $\delta = (\mathbf{v}^{(i-1)}\,,\,\mathbf{r}^{(k)}\,)$ $\quad (\,\delta = (\mathbf{v}^{(0)}\,,\,\mathbf{r}^{(1)}\,)\,)$

    $\mathbf{y} = \mathbf{y} - \delta\,\mathbf{v}^{(i-1)}$ $\quad (\,\mathbf{y} = \mathbf{y} - \delta\,\mathbf{v}^{(0)}\,)$

    $\mathbf{y} = \mathbf{y} \,/\, \|\,\mathbf{y}\,\|$

    end loop

    $\mathbf{v}^{(k-1)} = \mathbf{y}$ $\quad (\,\mathbf{v}^{(1)} = \mathbf{y}\,)$
11. Return to step #3.

*Comments:*

Step # 10 is not really required for small systems. It is intended for very large systems where the error can accumulate to significant proportions within a few iterations. Davidson's method is very sensitive to round-off errors. As a result, even for the smallest systems, the method will take a long time to converge if single-precision arithmetic is used, and convergence may not even occur. Double-precision is a minimum requirement for Davidson's method to perform successfully.

The algorithm steps given above will perform just slightly better than the Conjugate Gradient Method. However, steps can be taken to significantly improve the convergence rate of this algorithm. Even without the changes that will be proposed, it should be kept in mind that this system has the ability to solve a system of linear equations where the coefficient matrix is asymmetric. That in itself should be enough to justify its use.

**(v) Improving the Procedure:**

*(a) Reducing the Spectral Radius:*

It is a well known fact that the convergence rate is related to the spectral radius of the coefficient matrix. In other words, the value of the largest eigenvalue has an impact on the rate of convergence, the higher the value, the slower the rate. Therefore, if something can be done to a system of linear equations so that the spectral radius is reduced, then the convergence rate can be dramatically reduced.

The reduction of the spectral radius can be accomplished by multiplying the system of linear equations by the inverse of the diagonal matrix whose elements are the diagonal elements of the coefficient matrix.

Let D be the diagonal matrix whose elements contain the diagonal elements $A_{(i,i)}$ of the coefficient matrix.

$$D_{(i,j)} = A_{(i,j)} \quad \text{for } i = j$$

$$= 0 \quad \text{for } i \neq j.$$

The inverse of this matrix is simply the matrix defined by

$$D^{-1}{}_{(i,j)} = 1/A_{(i,j)} \quad \text{for } i = j$$

$$= 0 \quad \text{for } i \neq j.$$

If we now take the matrix $D^{-1}$ and multiply it by the linear system of equations

$$[D^{-1}][A][\mathbf{x}] = [D^{-1}][\mathbf{b}],$$

the solution of this system of equations is the same as the solution of the original system. At the same time, for a diagonally dominant system, the coefficient matrix $[D^{-1}][A]$ will produce a set of eigenvalues whose magnitudes are significantly less than those of the original coefficient matrix [A].

In other words

$$\lambda_{max}([D^{-1}][A]) < \lambda_{max}([A]).$$

The magnitude of this difference will depend on the level of diagonal dominance of the original coefficient matrix. Now, if an iterative procedure is used to solve the new system of linear equations, the required number of iterations will be less than if the procedure were being used on the original system of equations.

The improvement suggested above will generally reduce the number of iterations required to converge to the solution vector within the desired tolerance. However, it introduces a new problem of storing the new coefficient matrix, and the new constant vector. Although reducing the number of iterations is undoubtedly beneficial, having to store a modified coefficient matrix is not a desirable property – especially, when dealing with large systems of equations.

A practical means of implementing the benefits of solving the modified system of equations exists so that storing the new system is not necessary. Also, the

modification will add only *n* computations to each iteration. Considering the gain, this is not at all a significant addition to the computational requirements. The change is as follows:

Referring to step 9 of the computational steps outlined above, with a slight modification to the formula for **y**, the following is obtained

9. $\mathbf{y} = D^{-1}(A\mathbf{x}^{(k)} - \mathbf{b})$,

but this is equivalent to

9. $\mathbf{y} = ( Ax_i^{(k)} - b_i )/A_{(i,i)}$ for i = 1, n, and

$\mathbf{r}^{(k)} = \mathbf{y} / \| \mathbf{y} \|$ .

There is a slight difference between this particular implementation and the original suggestion of multiplying the entire system by the matrix $D^{-1}$ as far as the computational steps are concerned.

First, the way in which the computational steps are set up, the program will find the values of the vector $\alpha^{(k)}$ which correspond to a residual vector $\mathbf{r}^{(k)} = A\mathbf{x}^{(k)} - \mathbf{b}$ that is orthogonal to the space spanned by $\{V^{(k)} \in \mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(k)}\}$. However, when the residual vector is calculated using the new scheme, its direction will be calculated as $\mathbf{r}^{(k)} = D^{-1}( A\mathbf{x}^{(k)} - \mathbf{b} )$. This direction will not be orthogonal to the space spanned by $V^{(k)}$, however, this does not cause any problems since step # 10 contains an algorithm that will produce a vector orthonormal to $V^{(k)}$. As a result, the orthonormal base which emerges will be identical to the one that would be obtained had the original step # 9 been used to solve the system $D^{-1}A\mathbf{x} = D^{-1}\mathbf{b}$. The values of

the components of the vector $\alpha^{(k)}$ from these two methods would be different however, up to the final iteration, at which point they would be the same. The following section containing examples will illustrate that this is true.

Solving a number of systems using this new scheme and comparing the performance to that of the original system has consistently resulted in a significant increase in the convergence rate.

*(b) Condensing the orthonormal base*

It may have become apparent by now that after a number of iterations, we are left with the problem of solving a linear system of equations where the size of the system corresponds to the current iteration number. This does not appear to provide much benefit since the $k^{th}$ iteration requires solving k equations in k unknowns. Furthermore, it was suggested that a direct solver be used to evaluate the vector $\alpha^{(k)}$ at each iteration. This means that by the time the $k^{th}$ iteration has been completed, $k$ systems of linear equations have been solved, starting from a system of one equation in one unknown up to and including a system of $k$ equations in $k$ unknowns.

Without regard to any of the other computations in the rest of the program, there now appears to be a significant amount of effort expended in performing these computations alone. This would not be a problem for relatively small systems of equations, however, for large systems where there would be many iterations, this can lead to a significant slow down. There is a way to get around this problem, and that is

to condense the orthonormal base into a single vector after a preset number of iterations. This begins by defining

$$\mathbf{z} = \alpha_1^{(k)}\mathbf{v}^{(0)} + \alpha_2^{(k)}\mathbf{v}^{(1)} + \ldots + \alpha_m^{(k)}\mathbf{v}^{(k)}$$

where $k$ is the current iteration number, and $m$ is the preset number of iterations after which the condensation takes place. The vector $\mathbf{z}$ is really the current approximation to the solution $\mathbf{x}^{(k)}$. It is also a direction vector representing the orthonormal base spanned by $V^{(k)}$. The matrix V can now be cleared, and its first column would contain the vector $\mathbf{z}$. A number of other operations must also be carried out to reset all the appropriate matrices. These operations are outlined below in reference to the computational steps outlined above.

The matrix M which contains the transformed vectors $A\mathbf{v}^{(k)}$ must be reset so that it now contains a single vector. The following expression shows how this is to be accomplished:

$$[M^{(new)}]_{(n,1)} = [M^{(old)}]_{(n,k)}\,[\alpha^{(k)}]_{(k,1)}$$

$$= \alpha_1^{(k)}A\mathbf{v}^{(0)} + \alpha_2^{(k)}A\mathbf{v}^{(1)} + \ldots + \alpha_m^{(k)}A\mathbf{v}^{(k)} . \qquad (3 - 8)$$

A similar operation must take place for the matrix V,

$$[V^{(new)}]_{(n,1)} = [V^{(old)}]_{(n,k)}[\alpha^{(k)}]_{(k,1)} . \qquad (3 - 9)$$

The final set of computations must be carried out to condense the secondary system of linear equations $W\alpha = \mathbf{B}$. We wish to reduce each of the matrices W and $\mathbf{B}$ to a single element respectively. This can be accomplished by multiplying both sides of the

equation by the row vector $\alpha^T$. This multiplication results in a single dimensional matrix W, and a new vector **B** containing a single component. The new vector $\alpha$ will contain one component whose value is unity.

$$[W^{(new)}]_{(1,1)} = [\alpha^T]_{(1,k)} [W^{(old)}]_{(k,k)} [\alpha]_{(k,1)},$$

$$[\mathbf{B}^{(new)}]_{(1,1)} = [\alpha^T]_{(1,k)} [\mathbf{B}^{(old)}]_{(k,n)},$$

giving rise to the new system of equations

$$[W^{(new)}]_{(1,1)} [\alpha^{(new)}]_{(1,1)} = [\mathbf{B}^{(new)}] \qquad (3 - 10)$$

where $$[\alpha^{(new)}] = [\,1\,]. \qquad (3\ \ 11)$$

Care must be taken when implementing this new system. The matrices must be reset to their new values at the appropriate stage in the computational steps to avoid losing any relevant information. The final version of the computational steps given at the end of this section shows how this can be accomplished.

Another advantage appears from performing this condensation. Since the condensation takes place after a predetermined number of iterations, a fixed amount of storage space can be allotted to the matrices V, W, and M, as well as the vectors $\alpha$ and **B**. Without this condensation step, the sizes of these matrices could not be pre-assigned, and therefore would have to be allotted extremely large amounts of space to prevent any of them from being too small. For example, if the original system of equations were of size $n = 100$, and the preset iteration count at which to perform the condensation $m = 20$, then it would be known that the sizes of the matrices V, W, and

M would not exceed (100, 20), (20, 20) and (100, 20) respectively. Similarly, the vectors $\alpha$ and **B** could be set at (20, 1) for both vectors.

*(c) The improved Computational Procedure:*

Some new variables are introduced to keep track of the iteration count. The variable *it* keeps track of the iteration count, *itc* keeps track of the iteration count up to the condensation step after which it resets to 1, *m* is the preset number of iterations after which the condensation step takes place, and *flag* is the variable which is used to show whether condensation has taken place.

1. SET it = 0, and itc = 0.
2. Obtain an initial guess $\mathbf{x}^{(0)}$.
3. it = it + 1, and itc = itc + 1.
4. Normalize $\mathbf{x}^{(0)}$ to obtain $\mathbf{v}^{(0)}$ as the first search direction. The span of $V^{(1)}$ consists of the vector $\mathbf{v}^{(0)}$.
5. The vectors $\{\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots \mathbf{v}^{(k-1)}\}$ are stored in the matrix $V^{(k)}$.
6. Calculate and store the vectors $A\mathbf{v}^{(k)}$ as the columns of a matrix M, $A\mathbf{v}^{(0)}$ is stored in $M^{(1)}$ in this case.
7. Calculate the components of the matrix $W^{(k)}$ and the vector $\mathbf{B}^{(k)}$ from
   $$W^{(k)} = [V^T]^{(k)}[\,A\,][V\,]^{(k)} = [V^T]^{(k)}\,[M]^{(k)} \qquad \&$$
   $$\mathbf{B}^{(k)} = [V^T]^{(k)}[\,\mathbf{b}\,].$$
8. Using a direct solver such as Gauss-Jordan elimination, solve the following system of equations for $\alpha^{(k)}$
   $$W^{(k)}\,\alpha^{(k)} = \mathbf{B}^{(k)}.$$
9. flag = 0.

10. IF itc = m, THEN

    flag = itc

    $[M] = [M][\alpha^{(m)}]$

    $[W_{(1,1)}] = [\alpha^T][W][\alpha]$

    $[\mathbf{B}_{(1)}] = [\alpha^T][\mathbf{B}]$

    itc = 1

    END

    IF flag $\neq$ 0, THEN

    $A\mathbf{x}^{(k)} = 1.0 * [M]$

    Go to step #12

    END

11. Calculate the latest approximation to the constant vector using

    $A\mathbf{x}^{(k)} = [M^{(k)}]\,[\alpha^{(k)}]$

12. Check whether the convergence criteria has been satisfied using

    $\max |( Ax_i^{(k)} - b_i)/b_i| < \varepsilon$ for i = 1, 2, …, n where

    $\varepsilon$ is the convergence criteria,

    & flag = 0, to ensure that condensation is not taking place during this iteration

13. Form the new residual vector direction $\mathbf{r}^{(k)}$ from

    $\mathbf{y} = ( Ax_i^{(k)} - b_i )/A_{(i,i)}$ for i = 1, n, and

    $\mathbf{r}^{(k)} = \mathbf{y} / \| \mathbf{y} \|$ .

14. Calculate the vector $\mathbf{v}^{(k)}$ that is orthogonal to the base $V^{(k)}$ using the following algorithm for the Gram-Schmidt process:

    $\mathbf{y} = \mathbf{r}^{(k)}$

    for i = 1, k

    $\delta = (\mathbf{v}^{(i-1)}, \mathbf{r}^{(k)})$

    $\mathbf{y} = \mathbf{y} - \delta\, \mathbf{v}^{(i-1)}$

    $\mathbf{y} = \mathbf{y} / \| \mathbf{y} \|$

    end loop

    $\mathbf{v}^{(k-1)} = \mathbf{y}$

15. IF flag = 0 THEN

    Return to step #3

  ELSE

    $[V_{(i,\,1)}] = [V^{(m)}][\alpha^{(m)}]$,

    $[\alpha^{(1)}] = 1.0$

    Return to step #3

END

*(d) A Graphical description of the Process:*

This section will go through a step by step description of the process taking place. The description will refer to a number of accompanying figures.

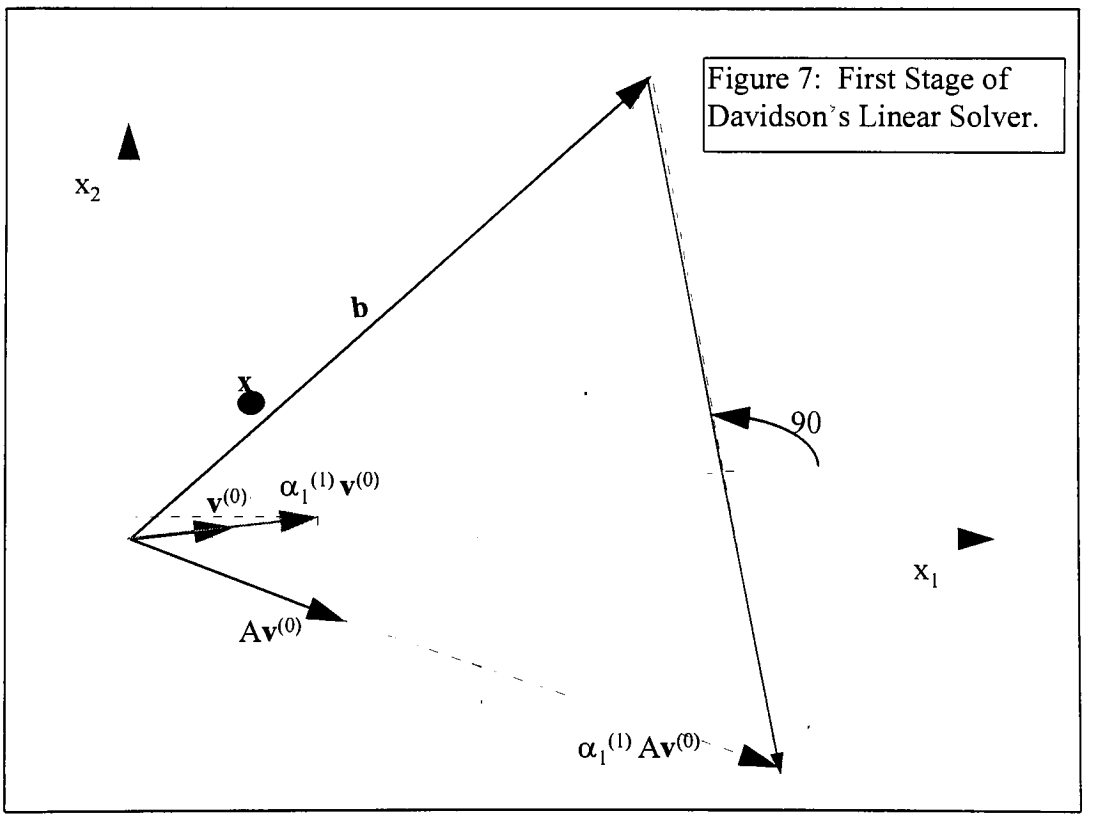



Figure 7: First Stage of Davidson's Linear Solver.

In Figure 7, **x** is the true solution of a system with $n = 2$, and the vector **b** is the constant vector. An arbitrary initial guess is given, and the guess vector is normalized giving us the vector $\mathbf{v}^{(0)}$. The transformed vector $A\mathbf{v}^{(0)}$ is also shown. A scalar multiple $\alpha_1^{(1)}$ will be calculated such that the residual vector $\mathbf{r}^{(1)}$ is orthogonal to the direction $\mathbf{v}^{(0)}$. This is done using eqn (3-2). The scalar product $\alpha_1^{(1)}\mathbf{v}^{(0)}$ is the first approximation to the solution **x**. Its projection onto the $x_1$ and $x_2$ axes are also shown in Figure 7. Notice that in the first iteration, the length of the residual vector has increased compared to the residual ($A\mathbf{v}^{(0)}$ - **b**). This appears somewhat counter-intuitive, but it is the best possible approximation that can be provided by this method along the direction $\mathbf{v}^{(0)}$.

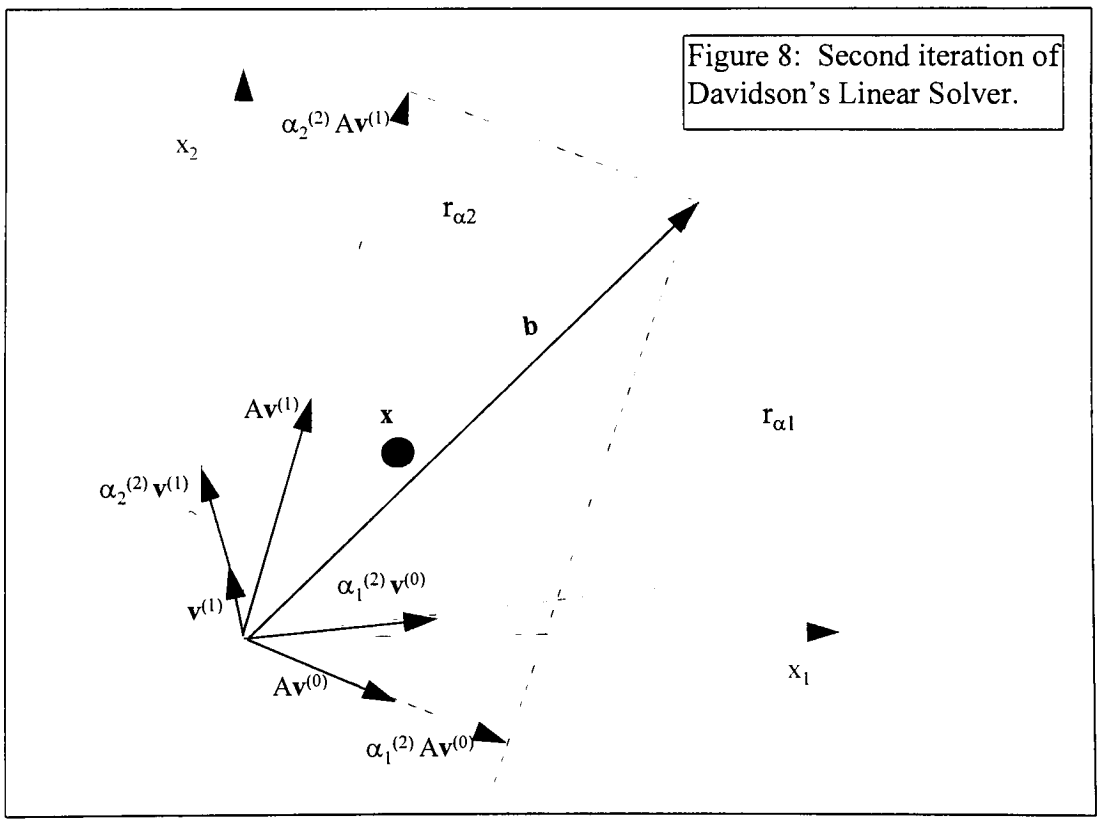



Figure 8: Second iteration of Davidson's Linear Solver.

In Figure 8, the second iteration of Davidson's Linear Solver can be seen. First an orthogonal direction is chosen using the Gram Schmidt Algorithm. This process is essentially building a new vector at each iteration that is orthogonal to all the vectors from previous iterations, i.e. the preceding orthonormal base. In this case the direction that is chosen is $\mathbf{v}^{(1)}$. Now two scalar multiples $\alpha_1$ and $\alpha_2$ are desired such that the projections of the residual vector ($\mathbf{b} - \alpha_1^{(2)}\mathbf{A}\mathbf{v}^{(0)} - \alpha_2^{(2)}\mathbf{A}\mathbf{v}^{(1)}$) onto the $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$ ($r_{\alpha 1}$ & $r_{\alpha 2}$ respectively) will be orthogonal to both of these directions simultaneously. This will be accomplished using either eqn (3-3) or (3-4) -- They are both the same.

At this point, the second approximation to the solution becomes

$$\mathbf{x}^{(2)} = \alpha_1^{(2)}\,\mathbf{v}^{(0)} - \alpha_2^{(2)}\,\mathbf{v}^{(1)}.$$

This is the true solution, and the sum of the vectors given in the expression above will point to the solution point **x**.

The problem outlined above can be viewed as a change of base (essentially a rotation of the coordinate system, where the problem is solved in a coordinate system that is different from the original. The new coordinate system defines its first direction based on an arbitrary initial guess, and all the later vectors are constructed using the projection of the constant vector onto the existing base. Thus, the method is searching for a subspace of the entire vector space in which the solution vector can be defined. It is not the smallest possible subspace in which to define the solution, but it

is still quite small. The smallest theoretical subspace in which the solution can be defined for a system of any size is $\mathbf{R}^2$, starting from an arbitrary initial guess.

For a larger system, if this method has not already converged, a new direction is chosen such that at the third iteration, the new direction must be orthogonal to both $\mathbf{v}^{(0)}$ and $\mathbf{v}^{(1)}$. Based on Figure 8, the next direction vector to include in the search $\mathbf{v}^{(2)}$ would travel either into or out of the plane of the paper. The search for the solution would then be conducted in the space spanned by $\mathbf{v}^{(0)}$, $\mathbf{v}^{(1)}$, and $\mathbf{v}^{(2)}$.

If the initial vector is chosen such that it points in the direction of the solution, with the magnitude not necessarily correct, Davidson's method converges in one iteration. This is an observation that was made by taking a system whose solution was known and multiplying that solution by an arbitrary scalar. The vector was then used as an initial guess for this method.

The same approach was then attempted for the Conjugate Gradient method, and it was found that there was no significant decrease in the number of iterations it took to converge. The explanation of this is simple. The Conjugate Gradient Method takes the residual of the initial guess as the direction along which to conduct the initial search. Davidson's method on the other hand, searches along the direction of the initial guess for the best approximation.

# IV. Conclusion:

## (i) Illustrative Examples

In the example that follows, a simple 10-by-10 coefficient matrix A was chosen which is symmetric and positive definite. A symmetric matrix is used to illustrate the difference in the performance of the methods. It should be kept in mind that Davidson's Method will work just as well for an asymmetric matrix, and no change needs to be made to the Final Algorithm of section III-(ii).

In the case of Davidson's method, the initial guess was calculated by dividing the elements of the constant vector **b** by the diagonal elements of the coefficient matrix A respectively. The following coefficient matrix and constant vector **b** were used:

| A | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 6 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 4 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 10 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 9 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 12 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 9 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 8 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 7 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 13 |

| b |
|---|
| 21 |
| 13 |
| 14 |
| 19 |
| 22 |
| 30 |
| 16 |
| 21 |
| 19 |
| 15 |

The results from Davidson's method are displayed in Tables 4-1 and 4-2. In Table 4-1, the system was solved using the Final Algorithm. In Table 4-2, the system of equations was multiplied by $D^{-1}$, giving ( $[D^{-1}A]\mathbf{x} = [D^{-1}]\mathbf{b}$ ), and the corresponding step in the Final Algorithm (step # 13 -- division by $A_{(i,\,i)}$) was omitted. It can be seen that

in both cases, the vectors spanning $V^{(k)}$ are identical, although the corresponding values of the vectors $\alpha^{(k)}$ are not – with the exception of the final vector, which corresponds to the solution.

The values of the quadratic function $F(\mathbf{v})$ are also different for the two cases, but this is obvious since the values were calculated using eqn 1-1 and two different coefficient matrices. In both cases, the process converged to the solution in exactly five iterations.

For the same matrix given above, the Conjugate Gradient method was carried out. The results are displayed in Table 4-3. It was found that for the same starting guess as was used for Davidson's method, the process required eight iterations to converge. For a 10-by-10 matrix, Davidson's method was still able to converge significantly faster than the Conjugate Gradient method

A further demonstration of the difference in performance is given in Table 4-4. For this table a simple Matlab program was used to create a symmetric matrix, in which the diagonal dominance can be varied as desired. The performance of the two methods are compared in this table.

## Solution of Ax = b

| IT | CONX |
|---|---|
| 1 | 0.2816120 |
| 2 | 0.0153111 |
| 3 | 0.0013374 |
| 4 | 0.0000428 |
| 5 | 0.0000014 |

| $\alpha^{(1)}$ | $\alpha^{(2)}$ | $\alpha^{(3)}$ | $\alpha^{(4)}$ | $\alpha^{(5)}$ |
|---|---|---|---|---|
| 3.8689 | 3.8551 | 3.8532 | 3.8529 | 3.8529 |
| 0 | -1.2354 | -1.2765 | -1.2767 | -1.2767 |
| 0 | 0 | -0.0868 | -0.0883 | -0.0883 |
| 0 | 0 | 0 | -0.0046 | -0.0047 |
| 0 | 0 | 0 | 0 | -0.0002 |

| Quadratic Function Value -- F(**x**) | | | | |
|---|---|---|---|---|
| -112.2721 | -117.0395 | -117.0613 | -117.0614 | -117.061 |

| Vectors spanning $V^{(k)}$ | | | | | Solution |
|---|---|---|---|---|---|
| $\mathbf{v}^{(0)}$ | $\mathbf{v}^{(1)}$ | $\mathbf{v}^{(2)}$ | $\mathbf{v}^{(3)}$ | $\mathbf{v}^{(4)}$ | **x** |
| 0.5050004 | -0.3574433 | -0.4517981 | 0.0006415 | 0.4526400 | 2.4418466 |
| 0.2605161 | 0.5179170 | -0.0839875 | -0.7519960 | -0.1823655 | 0.3534752 |
| 0.4208337 | 0.5096608 | 0.5240078 | 0.4248508 | 0.2747829 | 0.9224588 |
| 0.2284526 | 0.0195220 | -0.0900482 | 0.0428700 | -0.1081181 | 0.8630398 |
| 0.2939156 | -0.1719949 | 0.0634850 | 0.1166354 | -0.4738579 | 1.3459208 |
| 0.3005955 | -0.4670916 | 0.5666482 | -0.3592901 | -0.0372788 | 1.7061251 |
| 0.2137568 | 0.1955096 | -0.2461981 | -0.0549995 | 0.3131893 | 0.5959185 |
| 0.3156252 | -0.1400270 | -0.0129640 | 0.1481781 | -0.3537818 | 1.3953365 |
| 0.3263608 | -0.0213748 | -0.1123710 | 0.0166846 | -0.0340303 | 1.2945596 |
| 0.1387364 | 0.1944630 | -0.3280280 | 0.2901936 | -0.4736312 | 0.3139492 |

| $F(\mathbf{x}^{(5)})$ | -117.061388759 |
|---|---|

**Table 4-1: Davidson's Method**

## Solution of $[D^{-1}A]\ x = [D^{-1}]\ b$

| IT | CONX |
|---|---|
| 1 | 0.2576695 |
| 2 | 0.0210185 |
| 3 | 0.0013507 |
| 4 | 0.0000553 |
| 5 | 0.0000018 |

| $\alpha^{(1)}$ | $\alpha^{(2)}$ | $\alpha^{(3)}$ | $\alpha^{(4)}$ | $\alpha^{(5)}$ |
|---|---|---|---|---|
| 3.7966 | 3.8560 | 3.8530 | 3.8529 | 3.8529 |
| 0 | -1.2698 | -1.2766 | -1.2767 | -1.2767 |
| 0 | 0 | -0.088 | -0.0883 | -0.0883 |
| 0 | 0 | 0 | -0.0047 | -0.0047 |
| 0 | 0 | 0 | 0 | -0.0002 |

| Quadratic Function Value -- F(**x**) | | | | |
|---|---|---|---|---|
| -15.788 | -16.035 | -16.0225 | -16.0219 | -16.0218 |

| Vectors spanning $V^{(k)}$ | | | | | Solution |
|---|---|---|---|---|---|
| $\mathbf{v}^{(0)}$ | $\mathbf{v}^{(1)}$ | $\mathbf{v}^{(2)}$ | $\mathbf{v}^{(3)}$ | $\mathbf{v}^{(4)}$ | **x** |
| 0.5050004 | -0.3574433 | -0.4517981 | 0.0006415 | 0.4526400 | 2.4418456 |
| 0.2605161 | 0.5179170 | -0.0839875 | -0.7519960 | -0.1823655 | 0.3534753 |
| 0.4208337 | 0.5096608 | 0.5240078 | 0.4248508 | 0.2747829 | 0.9224582 |
| 0.2284526 | 0.0195220 | -0.0900482 | 0.0428700 | -0.1081181 | 0.8630398 |
| 0.2939156 | -0.1719949 | 0.0634850 | 0.1166354 | -0.4738579 | 1.3459213 |
| 0.3005955 | -0.4670916 | 0.5666482 | -0.3592901 | -0.0372788 | 1.7061250 |
| 0.2137568 | 0.1955096 | -0.2461981 | -0.0549995 | 0.3131893 | 0.5959179 |
| 0.3156252 | -0.1400270 | -0.0129640 | 0.1481781 | -0.3537818 | 1.3953368 |
| 0.3263608 | -0.0213748 | -0.1123710 | 0.0166846 | -0.0340303 | 1.2945594 |
| 0.1387364 | 0.1944630 | -0.3280280 | 0.2901936 | -0.4736312 | 0.3139498 |

| $F(\mathbf{x}^{(5)})$ | -16.021779419 |
|---|---|

**Table 4-2: Davidson's Method**

**Table 4-3: Conjugate Gradient Method**

| k | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|---|
| $e_{k-1}$ | ---------- | ---------- | 0.020302 | 0.1320 | 0.112813 | 0.154526 | 0.08906 | 0.064535 | 0.0521607 |
| $p^{(k)}$ | ---------- | -20.7820 | -3.5452 | -0.9905 | 0.1180 | 0.1880 | 0.0858 | 0.02405 | 0.004455 |
| | ---------- | -22.8154 | -3.8452 | -0.6177 | 0.4341 | 0.2296 | 0.0292 | -0.01080 | -0.005092 |
| | ---------- | -21.4820 | -5.3306 | -2.7566 | -0.9291 | -0.2819 | -0.0500 | -0.00583 | -0.000456 |
| | ---------- | -23.0820 | 1.2533 | 1.1017 | -0.3857 | 0.0235 | 0.0569 | -0.01701 | 0.002061 |
| | ---------- | -22.5376 | 0.2355 | 1.1882 | -0.0902 | -0.1005 | 0.0124 | 0.00765 | -0.002592 |
| | ---------- | -22.4820 | 4.1341 | -0.1979 | -0.2968 | 0.2385 | -0.0460 | 0.00464 | -0.000408 |
| | ---------- | -23.2042 | -0.1389 | 1.1763 | -0.0389 | -0.1079 | 0.0043 | 0.00924 | -0.002388 |
| | ---------- | -22.3570 | -0.9450 | 0.9786 | 0.2531 | -0.0934 | -0.0531 | 0.00682 | 0.002861 |
| | ---------- | -22.2677 | -2.1666 | 0.4432 | 0.4825 | 0.0458 | -0.0614 | -0.01491 | 0.001969 |
| | ---------- | -23.8282 | 4.9760 | -1.1112 | 0.3432 | -0.1423 | 0.0210 | -0.00204 | 0.000041 |
| $z^{(k)} = Ap^{(k)}$ | ---------- | -307.966 | -19.5532 | -4.7478 | 0.3624 | 0.7512 | 0.3422 | 0.09800 | 0.018268 |
| | ---------- | -338.915 | -24.5985 | -3.8745 | 2.0608 | 1.1472 | 0.1449 | -0.05216 | -0.025012 |
| | ---------- | -289.284 | -21.3644 | -9.0555 | -2.8970 | -0.8465 | -0.1509 | -0.01566 | -0.000920 |
| | ---------- | -432.576 | 5.9070 | 9.1290 | -3.5812 | 0.2105 | 0.5116 | -0.15132 | 0.019002 |
| | ---------- | -405.139 | -3.4885 | 8.7196 | -0.8311 | -0.8048 | 0.0979 | 0.06298 | -0.020286 |
| | ---------- | -472.140 | 40.1021 | -2.9624 | -3.3745 | 2.6228 | -0.5068 | 0.05287 | -0.004042 |
| | ---------- | -410.472 | -6.4836 | 8.6245 | -0.4209 | -0.8640 | 0.0336 | 0.07577 | -0.018658 |
| | ---------- | -381.337 | -11.9879 | 6.0646 | 1.6621 | -0.6547 | -0.3726 | 0.04956 | 0.020475 |
| | ---------- | -358.445 | -18.3723 | 1.8732 | 2.7855 | 0.2741 | -0.3695 | -0.08762 | 0.012260 |
| | ---------- | -510.776 | 54.3390 | -14.1203 | 4.0089 | -1.7082 | 0.2510 | -0.02269 | 0.000941 |
| $h = (z^{(k)}, p^{(k)})$ | ---------- | 88280.17 | 772.58 | 85.663 | 9.2440 | 1.7647 | 0.1427 | 0.00870 | 0.000430 |
| $q^{(k)}$ | ---------- | 0.05734 | 0.13302 | 0.15841 | 0.16561 | 0.13405 | 0.14767 | 0.15622 | 0.16498 |

**Table 4-3: Conjugate Gradient Method**

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| $x^{(k)}$ | 4.2000 | 3.0084 | 2.5368 | 2.3799 | 2.3994 | 2.4246 | 2.4373 | 2.44106 | 2.4417971 |
| | 2.1667 | 0.8584 | 0.3470 | 0.2491 | 0.3210 | 0.3518 | 0.3561 | 0.35440 | 0.3535556 |
| | 3.5000 | 2.2682 | 1.5592 | 1.1225 | 0.9686 | 0.9308 | 0.9234 | 0.92253 | 0.9224586 |
| | 1.9000 | 0.5765 | 0.7432 | 0.9177 | 0.8538 | 0.8570 | 0.8654 | 0.86272 | 0.8630623 |
| | 2.4444 | 1.1521 | 1.1835 | 1.3717 | 1.3568 | 1.3433 | 1.3451 | 1.34630 | 1.3458774 |
| | 2.5000 | 1.2109 | 1.7608 | 1.7294 | 1.6803 | 1.7123 | 1.7055 | 1.70619 | 1.7061235 |
| | 1.7778 | 0.4472 | 0.4288 | 0.6151 | 0.6087 | 0.5942 | 0.5948 | 0.59629 | 0.5958959 |
| | 2.6250 | 1.3430 | 1.2173 | 1.3724 | 1.4143 | 1.4018 | 1.3939 | 1.39499 | 1.3954640 |
| | 2.7143 | 1.4375 | 1.1493 | 1.2195 | 1.2994 | 1.3055 | 1.2964 | 1.29412 | 1.2944420 |
| | 1.1538 | -0.2125 | 0.4494 | 0.2734 | 0.3302 | 0.3112 | 0.3143 | 0.31394 | 0.3139486 |
| $r^{(k)}$ | 20.7820 | 3.1233 | 0.5223 | -0.2298 | -0.1698 | -0.0690 | -0.0185 | -0.00320 | -0.000187 |
| | 22.8154 | 3.3820 | 0.1100 | -0.5038 | -0.1625 | -0.0087 | 0.0127 | 0.00453 | 0.000403 |
| | 21.4820 | 4.8945 | 2.0527 | 0.6181 | 0.1384 | 0.0249 | 0.0026 | 0.00015 | 0.000001 |
| | 23.0820 | -1.7219 | -0.9362 | 0.5100 | -0.0831 | -0.0549 | 0.0207 | -0.00295 | 0.000186 |
| | 22.5376 | -0.6931 | -1.1571 | 0.2242 | 0.0866 | -0.0213 | -0.0068 | 0.00299 | -0.000356 |
| | 22.4820 | -4.5905 | 0.7438 | 0.2745 | -0.2844 | 0.0672 | -0.0076 | 0.00065 | -0.000016 |
| | 23.2042 | -0.3322 | -1.1946 | 0.1716 | 0.1019 | -0.0139 | -0.0090 | 0.00287 | -0.000208 |
| | 22.3570 | 0.4912 | -1.1034 | -0.1427 | 0.1325 | 0.0448 | -0.0102 | -0.00250 | 0.000873 |
| | 22.2677 | 1.7145 | -0.7293 | -0.4325 | 0.0288 | 0.0655 | 0.0109 | -0.00275 | -0.000723 |
| | 23.8282 | -5.4597 | 1.7683 | -0.4686 | 0.1953 | -0.0337 | 0.0034 | -0.00015 | 0.000008 |
| $\|\| r^{(k)} \|\|^2$ | 5061.98 | 102.7659 | 13.57012 | 1.530881 | 0.236561 | 0.021068 | 0.00136 | 7.09E-05 | 1.687E-06 |
| $F(x^{(k)})$ | 36.11939 | -109.0075 | -115.8423 | -116.9171 | -117.0439 | -117.0597 | -117.0613 | -117.0614 | -117.061 |

| | |
|---|---|
| $\|\| r^{(8)} \|\|^2$ | 1.68694916E-06 |
| $F(x^{(8)})$ | -117.061388628 |

The following program was used to construct the symmetric matrices that were solved in Table 4-4. The formulas used to calculate the values of the diagonal elements and the elements of the vector **b** are arbitrary. It merely ensures that the solution vector will not be an integer. It is just a convenient way to assemble large matrices. After running the program, the user is asked for the size desired, and then for the value that will be used for the off-diagonal elements.

```
clear;
ord = input('order of desired matrix = ');
y1 = input('value of off diagonal elements = ');
%
p = 11; q = 2; r = 1.2; s = 7; t = 1.5;u = 6;
di = (ord + q*(ord-1)^r);
dif = di/(p*ord / s);
%
for l = 1:ord
  for k = 1:l
    if k == l
      A(l,k) = di;
    else
      A(l,k) = y1;
      A(k,l) = A(l,k);
    end
  end
   di = di - dif;
    b(l) = s*(di^t) / u;
end
b = b';
```

For example, the following matrix is generated by using this program for ord = 5, with the value of the off diagonal elements as two.

**Coefficient Matrix [A]**

| | | | | |
|---|---|---|---|---|
| 15.5561 | 2 | 2 | 2 | 2 |
| 2 | 13.5762 | 2 | 2 | 2 |
| 2 | 2 | 11.5963 | 2 | 2 |
| 2 | 2 | 2 | 9.6165 | 2 |
| 2 | 2 | 2 | 2 | 7.6366 |

vector **b**

| |
|---|
| 58.3598 |
| 46.071 |
| 34.7913 |
| 24.6206 |
| 15.6963 |

Using similarly constructed matrices of different sizes, these systems of equations were solved using both Davidson's method and the Conjugate Gradient method. The table shows a summary of the performance of both methods for matrices of different sizes. Given in the table are the size of the matrix, the first and last diagonal elements of the coefficient matrix, the value of the off-diagonal elements, the value of the quadratic function defined by equation 1-1, and the number of iterations required to converge. The value of the quadratic function is given to illustrate that both methods converged to the same solution, producing quadratic function values that are essentially the same (well within six decimal places of each other).

Table 4-4 is followed by a summary of their performances, as well as some concluding remarks.

In the following table, n is the size of a square n-by-n matrix and c is the value of the off-diagonal elements. The column F(**x**) contains the value of the quadratic function. The table shows the number of iterations required by each method to converge to the value shown. The solution vector is not displayed.

| Performance Comparison: | | | | Conjugate Gradient | | Davidson | |
|---|---|---|---|---|---|---|---|
| n | c | A(1,1) | A(n,n) | F(x) | # iter | F(x) | # iter |
| 25 | 5 | 115.63 | 44.99 | -5.164804107807360E+04 | 11 | -5.164804107807630E+04 | 5 |
| 25 | 6 | 115.63 | 44.99 | -4.813254329786170E+04 | 11 | -4.813254329786540E+04 | 5 |
| 50 | 8 | 263.43 | 99.15 | -4.724964225687430E+05 | 12 | -4.724964225681190E+05 | 4 |
| 50 | 10 | 263.43 | 99.15 | -4.319308443921390E+05 | 12 | -4.319308443921450E+05 | 5 |
| 50 | 12 | 263.43 | 99.15 | -4.025749894361610E+05 | 12 | -4.025749894361660E+05 | 5 |
| 100 | 15 | 596.40 | 220.70 | -3.947662073458550E+06 | 14 | -3.947662073455790E+06 | 4 |
| 100 | 35 | 596.40 | 220.70 | -3.016404837142820E+06 | 15 | -3.016404837142080E+06 | 5 |
| 150 | 50 | 960.70 | 353.40 | -1.083410906665180E+07 | 16 | -1.083410906665080E+07 | 5 |
| 150 | 98 | 961.00 | 353.00 | -1.022353967442190E+07 | 17 | -1.022353967441990E+07 | 6 |
| 150 | 173 | 961.00 | 353.00 | -1.119892218088950E+07 | 20 | -1.119892218088840E+07 | 8 |
| 200 | 196 | 1347.00 | 494.00 | -2.719539763809190E+07 | 19 | -2.719539763808710E+07 | 7 |
| 200 | 391 | 1347.00 | 494.00 | -3.953225968473330E+07 | 32 | -3.953225968463580E+07 | 12 |
| 250 | 307 | 1751.00 | 641.00 | -5.987506882057660E+07 | 21 | -5.987506882057300E+07 | 8 |
| 250 | 571 | 1750.00 | 640.00 | -9.909334646511780E+07 | 44 | -9.909334646500030E+07 | 17 |

**Table 4-4: Summary of Performance for Conjugate Gradient Method and Davidson's Method**

### (ii) Concluding Remarks

The Conjugate Gradient Method can be a very effective tool for solving large systems of linear equations. However, its performance depends on a number of factors. First, if the coefficient matrix is asymmetric, then a huge number of additional computations must be carried out in order to make the method work. Secondly, if the matrix is not diagonally dominant then the convergence rate slows down drastically, and may not converge within $n$ iterations, where $n$ is the size of the coefficient matrix. When dealing with large matrices, this could become an enormous obstacle. However, there is no limit on the number of iterations that can be carried out, as long as sufficient memory and computational time is available.

On the other hand, Davidson's method does not require symmetry in the coefficient matrix. It also converges in significantly fewer iterations than the Conjugate Gradient Method. The convergence rate does tend to be affected somewhat by the conditioning of the matrix, but it still converges faster than the Conjugate Gradient Method. As can be seen in Table 4-4, in some instances the Conjugate Gradient method requires three times the number of iterations as Davidson's method. Furthermore, even for fairly large matrices ($n$ = 250) Davidson's method is seen to require less than ten iterations.

In order to truly appreciate the speed and efficiency of Davidson's method, it must be programmed and compiled in a programming language such as FORTRAN or C++ so that an executable file is obtained. The MATLAB programs that are supplied in the appendix tend to reduce the speed of the computations since they

consist of MATLAB commands, rather than executable programs. The number of iterations however, is not affected.

The difference in the performance of the two methods arises due to the fact that Davidson's method conducts its search in an increasing orthonormal subspace using the initial guess as the first search direction. It then searches for the values along these directions that will eliminate the projection of the residual vector onto that subspace. On the other hand, the Conjugate Gradient method is always constrained to search for the best approximation to the solution within a two dimensional plane. In other words, the freedom to move about in search of the best approximation is constrained, as opposed to Davidson's method where this freedom increases with each iteration – thus requiring fewer iterations overall.

# V. References


1. E. R. Davidson, *Journal of Computational Physics* **17**, 87-94 (1975)
2. Schwarz, H. R., Rutishauser H., and Steifel, E. (1973). Numerical Analysis of Symmetric Matrices. New Jersey: Prentice Hall.
3. Golub, G. H., and Van Loan, C. F. (1993). Matrix Computations. Johns Hopkins University Press.
4. M. R. Hestenes and W. Karush, *Journal of Research of the National Bureau of Standards* **47** (1951), 471.
5. A. Ghosh, *Evaluation of Direct and Iterative Methods for Large BEM Calculations*, presented at the 45th Anniversary Meeting of the SIAM, Stanford University, CA, July 14 – 18, 1997.


**Other References:**


1. Anton, H. and Rorres, C. (1994). Elementary Linear Algebra – Applications Version. New York: John Wiley and Sons, Inc.
2. Gerald, C. F.(1978). Applied Numerical Analysis. California: Addison-Wesley Publishing Company.
3. Householder, A. S. (1953). Principles of Numerical Analysis. New York: McGraw-Hill Book Company.
4. Hill, D. R. (1988). Experiments in Computational Matrix Algebra. New York: Random House/ Brikhauser Mathematics Series.
5. Hocking, J. G. (1970). Calculus with an Introduction to Linear Algebra. New York: Holt, Rinehart and Wisnton, Inc.
6. Davis, G. B. (1988). FORTRAN 77 – A Structured, Disciplined Style. New York: McGraw-Hill, Inc.
7. James, M. L., Smith, G. M., Wolford, J. C. (1993). Applied Numerical Methods for Digital Computation. New York: Harper Collins College Publishers.

*The following section contains programs for Conjugate Gradient, Davidson's Method and the original FORTRAN code from NASA Ames Research Center. Brief descriptions of the programs are given along with each.*

# VI. Appendix

## (i) Matlab Programs:

### Conjugate Gradient Method:

The program that was used is as given below. The variables used in the program are as follows:

| | |
|---|---|
| n | order of matrix ($A_{n \times n}$) |
| n1 | maximum number of iterations |
| b | elements of the constant vector |
| x | solution vector after maximum of n1 steps |
| q | corresponds to scalar multiple $q_k$ |
| e | corresponds to scalar multiple $e_k$ |
| ncg | current iteration number |

```
n = size(A,1);
n1 = 2*n
r = A*x - b;
p =-r;
%
% beginning of conjugate gradient iteration process.
%
for k = 1:n1
%
rr = 0;
 ncg = k - 1;
 for l = 1:n
  rr = rr + r(l)^2;
 end

if (rr < 1e-6)        % this is the tolerance criteria
  x,r
  break
 end

 if (k > 1)
  e(k-1) = rr/rr1;
  for l = 1:n
   p(l) = e(k-1)*p(l) - r(l);
  end
 end
```

```
  z = A*p;
h = 0;
  for l = 1:n
    h = h + p(l)*z(l);
  end
%
q(k) = rr/h;
  for l = 1:n
    x(l) = x(l) + q(k)*p(l);
    r(l) = r(l) + q(k)*z(l);
  end
%
end
```

**Davidson's Method:**

These are programs, subroutines and functions. In each case the name of the file to be used has been indicated towards the beginning in bold characters, followed by the '.m' extension. This is the program that was used to compare performances of the two methods.

Before beginning the sequence of programs the coefficient matrix A and the constant vector **b** must be input. The next step is to run the file named *start.m*. A number of buffer matrices have been defined in the program. Apart from that, the sequence of steps corresponds to those given in the Final Algorithm. The variables have also been named similar to the ones that were defined in the algorithm.

There are two types of files that are defined here. One is the command file which simply carries out a sequence of MATLAB commands in the order in which they are typed on the file. The other type of file is the function file, which gives an output based on variables that are input. The variables internal to the function file are local only, and cannot be accessed once the function file has been run. The command file makes use of global variables, and any variables that are assigned in a command file are treated as global variables by default.

The following list provides a brief description of the files that are given below:

| | |
|---|---|
| **start.m** | initializes variables for the process |
| **davidson.m** | program for method based on Final Algorithm |
| **condense.m** | condenses orthonormal base and re-initializes variables |
| **endfile.m** | termination file for this program |
| **GS.m** | Gaussian elimination – function file |
| **nv.m** | normalizes the input vector – function file |
| **setzero.m** | sets the components of a vector to zero – function file. |

```
% start.m: initialization file for davidson's method
%
% clear
D = diag(A);
matdim = size(b,1);
it = 0; itc = 0;
solres = 0.000005;
maxit = 200;
buf2 = b./D;
davidson
```

```
%  davidson.m:  Davidson's method -- runs off the file 'start.m'
%  where the initialization of variables takes place.
%
itc = itc + 1;
it  = it + 1;
buf2 = normal(buf2);
buf3 = A*buf2;
for k = 1:matdim
  M(k, itc) = buf3(k);
  V(k, itc) = buf2(k);
end
for k = 1:itc
  buf = V(:, k);
  buf2 = M(:, itc);
%----- formation of coefficient matrix W -----
  W(k, itc) = dot(buf, buf2);
  if k ~= itc
    buf = M(:, k);
    buf2 = V(:, itc);
    W(itc, k) = dot(buf, buf2);
  end
end
buf2 = V(:,itc);
%----- formation of vector B from inner product (b,v) -----
B(itc) = dot(b, buf2);
%
%----- calculation of alpha using Gaussian elimination -----
al = GS(W, B, itc);
alitc = al(itc);
flag = 0;
%
%----- check whether condensation should take place -----
if itc == 20
  condense;
end
buf2 = setzero(buf2);
for k = 1:itc
  buf = M(:,k);
  ali = al(k);
  if flag ~= 0
    ali = 1.0;
  end
  buf2 = buf2 + ali*buf;
end
conx = 0;
```

```
for k = 1:matdim
  q = abs((buf2(k) - b(k))/b(k));
  conx = max(abs(conx), abs(q));
end
buf2 = buf2 - b;
%-----record of iterations, current alpha value, and -----
%-----variable for checking convergence.      -----
record = [record; it, alitc, conx];
%
noconv = 0;
%-----check whether convergence criteria satisfied -----
if (conx < solres) & (flag == 0)
  endfile
end
%
noconv = 1;
if it == maxit
  endfile
end
buf2 = buf2./D;
buf2 = normal(buf2);
%-----orthonormalization step -----
lp = max(flag, itc);
for k = 1:lp
  buf = V(:,k);
  q = dot(buf, buf2);
  buf2 = buf2 - q*buf;
  buf2 = normal(buf2);
end
if flag == 0
  davidson
end
buf3 = setzero(buf3);
for k = 1:flag
  buf = V(:,k);
  ali = al(k);
  buf3 = buf3 + ali*buf3;
end
V(:,1) = buf3;
clear al;
al(1) = 1.0;
davidson
return
```

```
%  condense.m:  This has been set up as a command file
%  instead of a function file because there are several
%  variables which need to be changed. At the end of this
%  subroutine, it will return to davidson.m
%
flag = itc;
buf2 = setzero(buf2);
for k = 1:itc
  buf = M(:,k);
  ali = al(k);
  buf2 = buf2 + ali*buf;
end
M(:,1) = buf2;
q = 0.0;
y = 0.0;
for k = 1:itc
  for l = 1:itc
    q = q + W(k,l)*al(k)*al(l);
  end
  y = y + B(k)*al(k);
end
clear W;
W(1,1) = q;
clear B;
B(1) = y;
itc = 1;
return

function gout = GS(C, G, nitc)
%  GS.m is a standard Gaussian elimination scheme. It
%  requires the matrices buf1 and al, as well as the
%  itc count.  It does not perform pivoting as of right
%  now.
%
for k = 1:nitc
  fx = 1.0/C(k,k);
  G(k) = G(k)*fx;
  for m = k:nitc
    C(k,m) = fx*C(k,m);
  end
%
  for l = 1:nitc
    if k ~= l
      y = C(l,k);
      G(l) = G(l) - G(k)*y;
```

```
    for n = k:nitc
      C(l,n) = C(l,n) - y*C(k,n);
    end
   end
  end
end
gout = G;
return

function nv = normal(vector)
% normal.m:  this function will take a vector input and normalize
% it. The returned variable nv will have to be renamed.
%
 d = norm(vector);
 nv = vector/d;
return

function z = setzero(mat)
% setzero.m: sets the values of an input vector to zero
% the input vector must be in column form
%
nn = size(mat,1);
for k = 1:nn
 mat(k) = 0;
end
z = mat;
```

```
%  endfile.m: It is called when either the threshold
%  for convergence has been met or the maximum number
%  of iterations has been exceeded.It will not return
%  to the calling program, and it will stop here
%
buf = setzero(buf);
for k = 1:itc
  buf2 = V(:,k);
  alil = al(k);
  buf = buf + alil*buf2;
end
%disp('     IT      ALITC      CONX')
%disp('     ----     ------     ------')
%disp(record)
%Fxrec(it)
it
format long
Fx = 0.5*dot(A*buf, buf) - dot(b, buf)
format short
% solution = buf
if noconv == 1
  disp('-----------NO CONVERGENCE----------')
end
break
```

### (ii) Fortran Program:

Part of PMARC program extracted from a program written at NASA Ames Research Center. This is the program that was studied to develop the interpretation in section **IV**. Based on the interpretation, the algorithm for Davidson's Linear Solver was developed.

```
      PROGRAM DOUBLET
      C-------------------------------------------------------------------------
      C
          INCLUDE 'PARAM.DAT'
          INCLUDE 'COMMON.F'
      C
            OPEN(UNIT=2, FILE='A.dat',STATUS='OLD')
            OPEN(UNIT=3, FILE='B.dat',STATUS='OLD')
      C
            OPEN(UNIT=16, FILE='DATA6.dat',STATUS='NEW')
            OPEN(UNIT=20, FORM='UNFORMATTED',STATUS='UNKNOWN')
      C
            READ(2,*)((DUBICWW(I,J),J=1,NSPDIM),I=1,NSPDIM)
            READ(3,*)(RHSV(I),I=1,NSPDIM)
            CLOSE(2)
            CLOSE(3)
      C
            DO 2 I = 1,NSPDIM
       2    DIAG(I) = DUBICWW(I,I)
      C
      C  Rewind all scratch file 20 and assign unit number
      C
          IMU = 20
          REWIND IMU
      C
      C  CHECK TO MAKE SURE THAT IF INRAM IS NOT 1, THAT IT
      C  IS SET EQUAL TO NSPDIM
      C
          IF(INRAM.NE.1)THEN
           IF(INRAM.NE.NSPDIM)THEN
            WRITE(16,601)
            STOP
           ENDIF
          ENDIF
      C
      C  START THE TIME STEP LOOP
      C
          TSTIME = 0.0
      C
      C  WRITE INPUT DATA TO OUTPUT FILE
      C
          WRITE(16,603)
      C
        603 FORMAT(1X,10(/),31X,57('*')//
          +      54X,'PROGRAM PMARC'/
```

```
     +     45X,'Release version 12.21:  03/04/94'//
     +     51X,'MATRIX SOLVER EXTRACTED FROM PMARC'/)
      WRITE(24,*)((DUBICWW(I,J),J=1,NSPDIM),I=1,NSPDIM)
C
       CALL SOLVER
         OPEN(UNIT=78, FILE='solv2.out', STATUS ='NEW')
         WRITE(78,*) (DUB(I),I=1,NSPDIM)
C
C  CLOSE AND DELETE THE SCRATCH FILES
C
       CLOSE(UNIT=20,STATUS='DELETE')
C
      STOP
  600 FORMAT(//1X,'TIME STEP',I4)
  601 FORMAT(//1X,'PARAMETER INRAM NOT SET TO 1 OR NSPDIM IN PMARC'/
     +     1X,'SOURCE CODE. RESET THIS PARAMETER, RECOMPILE CODE'/
     +     1X,'AND TRY AGAIN.')
      END
C
*DECK SOLVER
      SUBROUTINE SOLVER
C
C-------------------------------------------------------------------------------
C
      INCLUDE 'PARAM.DAT'
      INCLUDE 'COMMON.F'
C
C  UPDATE THE PDUB ARRAY SO THAT IT ALWAYS HOLDS THE PREVIOUS
STEP'S DOUBLET
C  SOLUTION
C
         ITSTEP=0
         NPAN = NSPDIM
C
      DO 10 I=1,NPAN
       IF(ITSTEP.EQ.0)THEN
         PDUB(I) = RHSV(I)
       ELSE
         PDUB(I) = DUB(I)
         ENDIF
   10 CONTINUE
      CALL LINEQ(IT)
         WRITE(6,555) IT
  555  FORMAT(1X,'NUMBER OF ITERATIONS    = 'I5)
      WRITE(16,600)IT
C
      RETURN
  600 FORMAT(1X,'NUMBER OF SOLVER ITERATIONS = ',I4)
      END
      SUBROUTINE LINEQ(IT)
C
C    PROGRAM TO SOLVE LINEAR EQUATIONS (BASED ON: J. COMP. PHISICS,
C    "THE ITERATIVE CALCULATION OF....",  17. PP. 87-94, 1975)
C    FOR INFORMATION: CALL CHARLEY BAUSCHLICHER (415) 694-6231
```

```
C
      INCLUDE 'PARAM.DAT'
      INCLUDE 'COMMON.F'
C
      DIMENSION V(NSPDIM,20), W(NSPDIM,20),A(20,20),AL(20),
     + BUF(NSPDIM), GG(20), BUF1(400), BUF2(NSPDIM), BUF3(NSPDIM)
C
      IT = 0
         NPAN = NSPDIM
         SOLRES = 0.000005
         MAXIT = 200
      THRESH = SOLRES
      MATDIM = NPAN
      IMS = 0
C
C  INITIAL GUESS FOR STARTING SOLUTION VECTOR
C
      IF(ITSTEP.GT.0)THEN
       DO 10 I=1,MATDIM
        BUF2(I) = PDUB(I)
   10  CONTINUE
       GO TO 800
      ENDIF
      DO 20 I=1,MATDIM
       AHNN = DIAG(I)
       IF(ABS(AHNN).LT.1.E-7)AHNN = 1.0E-7
       BUF2(I) = RHSV(I)/AHNN
  20 CONTINUE
 800 CONTINUE
      WRITE(16,600)
      WRITE(16,601)
         WRITE(6,899)
 899   FORMAT(1X,'SOLUTION ITERATION HISTORY')
 810 CONTINUE
      IMS = IMS + 1
      CALL NORMAL(BUF2,MATDIM)
      IT = IT + 1
      CALL FRMAB(BUF2,BUF3,MATDIM,IMU)
C
      DO 30 I=1,MATDIM
       W(I,IMS) = BUF3(I)
       V(I,IMS) = BUF2(I)
   30 CONTINUE
      DO 40 I=1,IMS
       DO 50 J=1,MATDIM
        BUF(J) = V(J,I)
        BUF2(J) = W(J,IMS)
   50  CONTINUE
       A(I,IMS) = SDOT(MATDIM,BUF,1,BUF2,1)
       IF(I.EQ.IMS)GO TO 40
       DO 60 J=1,MATDIM
        BUF(J) = W(J,I)
        BUF2(J) = V(J,IMS)
   60  CONTINUE
```

```
      A(IMS,I) = SDOT(MATDIM,BUF,1,BUF2,1)
   40 CONTINUE
C
      DO 70 I=1,MATDIM
       BUF2(I) = V(I,IMS)
   70 CONTINUE
      GG(IMS) = SDOT(MATDIM,RHSV,1,BUF2,1)
      IQ = 0
      DO 80 I=1,IMS
       DO 90 J=1,IMS
        IQ = IQ + 1
        BUF1(IQ) = A(J,I)
   90  CONTINUE
       AL(I) = GG(I)
   80 CONTINUE
C
CALL GSS(BUF1,AL,IMS,IER)
C
IF(IER.EQ.1)THEN
      STOP
     ENDIF
     CONY = ABS(AL(IMS))
     WRITE(24,*)('CONY = ',CONY)
     IFOLD = 0
     IF(IMS.EQ.20)THEN
      IFOLD = IMS
      CALL ZERO(BUF2,MATDIM)
      DO 100 I=1,IMS
       DO 110 J=1,MATDIM
        BUF(J) = W(J,I)
  110   CONTINUE
       ALI = AL(I)
       CALL SAXPY(MATDIM,ALI,BUF,1,BUF2,1)
  100  CONTINUE
DO 120 J=1,MATDIM
       W(J,1) = BUF2(J)
  120  CONTINUE
      X = 0.0
      Y = 0.0
      DO 130 I=1,IMS
       DO 140 J=1,IMS
        X = X + A(I,J) * AL(I) * AL(J)
  140   CONTINUE
       Y = Y + GG(I) * AL(I)
  130  CONTINUE
      A(1,1) = X
      GG(1) = Y
      IMS = 1
     ENDIF
     CALL ZERO(BUF2,MATDIM)
     DO 150 I=1,IMS
      DO 160 J=1,MATDIM
       BUF(J) = W(J,I)
  160  CONTINUE
```

```
      ALI = AL(I)
      IF(IFOLD.NE.0)ALI = 1.0
      CALL SAXPY(MATDIM,ALI,BUF,1,BUF2,1)
  150 CONTINUE
    CONX = 0.0
    IPANEL = 1
    DO 170 I=1,MATDIM
      IF(ABS(RHSV(I)).LT.1.E-7) GO TO 820
      Q = ABS((BUF2(I) - RHSV(I))/RHSV(I))
      IF(CONX.LT.Q)THEN
       IPANEL = I
      ENDIF
      CONX = AMAX1(CONX,Q)
  820  CONTINUE
      BUF2(I) = BUF2(I) - RHSV(I)
  170 CONTINUE
    WRITE(16,602)IT,CONY,CONX,IPANEL
C
NOCONV = 0
    IF(CONX.LT.THRESH.AND.IFOLD.EQ.0)GO TO 830
    NOCONV = 1
    IF(IT.EQ.MAXIT) GO TO 830
    DO 180 I=1,MATDIM
      AHNN = DIAG(I)
      IF(ABS(AHNN).LT.1.0E-7)AHNN = 1.0E-7
      BUF2(I) = BUF2(I)/AHNN
  180 CONTINUE
    CALL NORMAL(BUF2,MATDIM)
    LP = MAX0(IFOLD,IMS)
    DO 190 I=1,LP
      DO 200 J=1,MATDIM
       BUF(J) = V(J,I)
  200  CONTINUE
      X = SDOT(MATDIM,BUF,1,BUF2,1)
      CALL SAXPY(MATDIM,-X,BUF,1,BUF2,1)
      CALL NORMAL(BUF2,MATDIM)
  190 CONTINUE
    IF(IFOLD.EQ.0)GO TO 810
    CALL ZERO(BUF3,MATDIM)
    DO 210 I=1,IFOLD
      DO 220 J=1,MATDIM
       BUF(J) = V(J,I)
  220  CONTINUE
      ALI = AL(I)
      CALL SAXPY(MATDIM,ALI,BUF,1,BUF3,1)
  210 CONTINUE
    DO 230 I=1,MATDIM
      V(I,1) = BUF3(I)
  230 CONTINUE
    AL(1) = 1.0
    GO TO 810
  830 CONTINUE
    CALL ZERO(BUF,MATDIM)
    DO 240 I=1,IMS
```

```
      DO 250 J=1,MATDIM
       BUF2(J) = V(J,I)
  250  CONTINUE
      ALIL = AL(I)
      CALL SAXPY(MATDIM,ALIL,BUF2,1,BUF,1)
  240 CONTINUE
     DO 260 I=1,MATDIM
      DUB(I) = BUF(I)
  260 CONTINUE
     IF(NOCONV.EQ.1) THEN
        WRITE(16,603)
        END IF
  600 FORMAT(1H1)
  601 FORMAT(1X,'SOLUTION ITERATION HISTORY'/)
  602 FORMAT(' IT=',I5,' AL(I) ',F15.8,' HV-G ',F15.8,' PANEL = ',I5)
  603 FORMAT(//'------------NO CONVERGENCE-----------')
     RETURN
     END

     SUBROUTINE FRMAB(A,B,MATDIM,IRAWM)
C
     INCLUDE 'PARAM.DAT'
     INCLUDE 'COMMON.F'
C
     DIMENSION A(MATDIM),B(MATDIM)
     REWIND IRAWM
     NPAN = MATDIM
     DO 10 I=1,NPAN
      II = I
      B(I) = 0.0
      IF(INRAM.EQ.1)THEN
       READ(IRAWM)(DUBICWW(INRAM,J),J=1,NPAN)
       II = 1
      ENDIF
      DO 20 J=1,NPAN
       B(I) = B(I) + A(J) * DUBICWW(II,J)
   20  CONTINUE
   10 CONTINUE
     RETURN
     END

     SUBROUTINE ZERO(A,LEN)
     DIMENSION A(LEN)
     DO 10 I=1,LEN
      A(I) = 0.0
   10 CONTINUE
     RETURN
     END

     SUBROUTINE NORMAL(A,LEN)
     DIMENSION A(LEN)
     X = 0.0
     DO 10 I=1,LEN
      X = X + A(I) * A(I)
```

```
 10 CONTINUE
    X = 1.0/SQRT(X)
    DO 20 I=1,LEN
      A(I) = A(I) * X
 20 CONTINUE
    RETURN,  END

SUBROUTINE GSS(B,G,NMIX,IER)
    DIMENSION B(NMIX,NMIX),G(NMIX)
C
C     OPEN(UNIT=28,FILE='BSOL.DAT',STATUS='NEW')
C
    WRITE(28,*)('MATRIX B (SENT AS BUF1), FOR IMS =',NMIX)
    WRITE(28,*)((B(I,J),I=1,NMIX),J=1,NMIX)
C
    DATA ZERO1/1.0E-16/
    IER = 0
    DO 10 I=1,NMIX
      IF(ABS(B(I,I)).LT.ZERO1)GO TO 800
      FX = 1./B(I,I)
      GO TO 810
 800  CONTINUE
      IF(I.EQ.NMIX)GO TO 820
      I1 = I + 1
C  PIVOT SECTION
      DO 20 J=I1,NMIX
        IF(ABS(B(J,I)).LT.ZERO1)GO TO 20
        FX = 1./B(J,I)
        DO 30 L=I,NMIX
          TEMP = B(J,L)
          B(J,L) = B(I,L)
          B(I,L) = TEMP
  30    CONTINUE
        TMP = G(J)
        G(J) = G(I)
        G(I) = TMP
        GO TO 810
  20  CONTINUE
      GO TO 820
 810  CONTINUE
      G(I) = G(I) * FX
      DO 40 J=I,NMIX
        B(I,J) = B(I,J) * FX
  40  CONTINUE
      DO 50 J=1,NMIX
        IF(I.EQ.J)GO TO 50
        Y = B(J,I)
        G(J) = G(J) - G(I) * Y
        DO 60 K=I,NMIX
          B(J,K) = B(J,K) - Y * B(I,K)
  60    CONTINUE
  50  CONTINUE
  10 CONTINUE
C
```

```
      WRITE(28,*)('G(I) =',(G(I),I=I,NMIX))
C
      RETURN
  820 CONTINUE
      WRITE(16,600)
  600 FORMAT('  ABORT INVERT  SINGULAR MATRIX ')
      IER = 1
      RETURN
      END

      SUBROUTINE SAXPY(N,SA,SX,INCX,SY,INCY)
C
C     CONSTANT TIMES A VECTOR PLUS A VECTOR.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DIMENSION SX(N),SY(N)
C
      IF(N.LE.0)RETURN
      IF(SA.EQ.0.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C     CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C     NOT EQUAL TO 1
C
      IX=1
      IY=1
      IF(INCX.LT.0)IX=(-N+1)*INCX+1
      IF(INCY.LT.0)IY=(-N+1)*INCY+1
      DO 10 I=1,N
        SY(IY)=SY(IY)+SA*SX(IX)
        IX=IX+INCX
        IY=IY+INCY
   10 CONTINUE
      RETURN
C
C     CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C     CLEAN-UP LOOP
C
   20 M=MOD(N,4)
      IF(M.EQ.0)GO TO 40
      DO 30 I=1,M
        SY(I)=SY(I)+SA*SX(I)
   30 CONTINUE
      IF(N.LT.4)RETURN
   40 MP1=M+1
      DO 50 I=MP1,N,4
        SY(I)=SY(I)+SA*SX(I)
        SY(I+1)=SY(I+1)+SA*SX(I+1)
        SY(I+2)=SY(I+2)+SA*SX(I+2)
        SY(I+3)=SY(I+3)+SA*SX(I+3)
   50 CONTINUE
C
```

```
      WRITE(24,*)('SY =',(SY(I),I=1,N))
C
      RETURN
      END

      REAL FUNCTION SDOT(N,SX,INCX,SY,INCY)
C
C     FORMS THE DOT PRODUCT OF TWO VECTORS.
C     USES UNROLLED LOOPS FOR INCREMENTS EQUAL TO ONE.
C     JACK DONGARRA, LINPACK, 3/11/78.
C
      DIMENSION SX(N),SY(N)
C
      STEMP=0.0E0
      SDOT=0.0E0
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1.AND.INCY.EQ.1)GO TO 20
C
C        CODE FOR UNEQUAL INCREMENTS OR EQUAL INCREMENTS
C        NOT EQUAL TO ONE.
C
      IX=1
      IY=1
      IF(INCX.LT.0)IX=(-N+1)*INCX+1
      IF(INCY.LT.0)IY=(-N+1)*INCY+1
      DO 10 I=1,N
        STEMP=STEMP+SX(IX)*SY(IY)
        IX=IX+INCX
        IY=IY+INCY
   10 CONTINUE
      SDOT=STEMP
      RETURN
C
C     CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C     CLEAN-UP LOOP
C
   20 M=MOD(N,5)
      IF(M.EQ.0)GO TO 40
      DO 30 I=1,M
        STEMP=STEMP+SX(I)*SY(I)
   30 CONTINUE
      IF(N.LT.5)GO TO 60
   40 MP1=M+1
      DO 50 I=MP1,N,5
        STEMP=STEMP+SX(I)*SY(I)+SX(I+1)*SY(I+1)+
     +     SX(I+2)*SY(I+2)+SX(I+3)*SY(I+3)+SX(I+4)*SY(I+4)
   50 CONTINUE
   60 SDOT=STEMP
      RETURN
      END
```